(1+1) Genetic Programming With Functionally Complete Instruction Sets Can Evolve Boolean Conjunctions and Disjunctions with Arbitrarily Small Error

Benjamin Doerr[1]**, Andrei Lissovoi**[2]**, and Pietro S. Oliveto**[3]

[1]doerr@lix.polytechnique.fr, Laboratoire d'Informatique (LIX), École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France
[2]a.lissovoi@sheffield.ac.uk, Department of Computer Science, University of Sheffield, UK
[3]p.oliveto@sheffield.ac.uk, Department of Computer Science, University of Sheffield, UK

Abstract

Recently it has been proven that simple GP systems can efficiently evolve a conjunction of n variables if they are equipped with the minimal required components. In this paper, we make a considerable step forward by analysing the behaviour and performance of a GP system for evolving a Boolean conjunction or disjunction of n variables using a complete function set that allows the expression of any Boolean function of up to n variables. First we rigorously prove that a GP system using the complete truth table to evaluate the program quality, and equipped with both the AND and OR operators and positive literals, evolves the exact target function in $O(\ell n \log^2 n)$ iterations in expectation, where $\ell \geq n$ is a limit on the size of any accepted tree. Additionally, we show that when a polynomial sample of possible inputs is used to evaluate the solution quality, conjunctions or disjunctions with any polynomially small generalisation error can be evolved with probability $1 - O(\log^2(n)/n)$. The latter result also holds if GP uses AND, OR and positive and negated literals, thus has the power to express any Boolean function of n distinct variables. To prove our results we introduce a super-multiplicative drift theorem that gives significantly stronger runtime bounds when the expected progress is only slightly super-linear in the distance from the optimum.

1 Introduction

Genetic Programming (GP) uses principles of Darwinian evolution to evolve computer programs with some desired functionality. The most popular and well-known GP approach, pioneered by Koza (1992), represents programs using syntax trees. It uses genetic variation operators to search through the space of programs composed of the available components, favouring ones which exhibit better behaviour on a wide variety of possible inputs. In this setting, the quality of a program is evaluated by comparing its outputs for varying inputs to those of the target function.

Despite the many examples of successful applications of GP (see e.g., Koza, 2010; Liu and Shao, 2013; Bartoli et al., 2014; Moore et al., 2018; Miranda et al., 2019; Lynch et al., 2019; Vu et al., 2019), our understanding of its behaviour and performance is limited.

The few available theoretical analyses of GP have followed the very successful path used in the analysis of traditional evolutionary algorithms (EAs) for function optimisation that initially considered simplified EAs such as the (1+1) evolutionary algorithm (Droste et al., 2002), and has progressively allowed the analysis of realistic EAs using populations and crossover (see, e.g., Jansen et al., 2005; Witt, 2006; Doerr et al., 2012a; Dang et al., 2018; Huang et al., 2019; Corus and Oliveto, 2020; Dang et al., 2021; Sutton, 2021; Corus et al., 2021; Zheng et al., 2022). Similarly simplified GP systems have been considered that use a tree-based mutation operator called HVL-Prime (an adaptation of the hierarchical variable length mutation proposed by O'Reilly and Oppacher, 1996) to evolve a single program. In this paper we consider RLS-GP (Durrett et al., 2011), which analogously to the *randomised local search* algorithm (see, e.g., Neumann and Wegener, 2007), performs a single local mutation before evaluating the fitness, differently to the $(1 + 1)$ GP which can perform a larger number of local changes in a single iteration, akin to the $(1 + 1)$ evolutionary algorithm.

Initial works analysed the behaviour and performance of these algorithms for the evolution of non-executable tree structures rather than the evolution of computer programs where the fitness function is an estimate of how well the the candidate solution matches the behaviour of the target program on a training set of inputs (Durrett et al.,

2011; Kötzing et al., 2014; Doerr et al., 2020; Lissovoi and Oliveto, 2020). Only recently the performance of simple GP systems has been analysed for the evolution of executable functions. It has been proven that Boolean conjunctions of n variables can be evolved by RLS-GP and (1+1) GP algorithms in an expected polynomial number of iterations (Mambrini and Oliveto, 2016; Lissovoi and Oliveto, 2019). Programs equivalent to the target conjunction can be evolved when the complete truth table (i.e. the set of all 2^n possible inputs) is used to evaluate the program quality. When the solution quality is evaluated by sampling a polynomial number of inputs uniformly at random from the complete truth table in each iteration (i.e. employing Dynamic Subset Selection to limit the total computational effort as suggested by Gathercole and Ross, 1994) the evolved programs are found to generalise well i.e., they are incorrect only on an arbitrarily small polynomial fraction of the 2^n possible inputs.

While these results are promising, the considered GP systems were considerably different from those used in practice. In particular, they were required to evolve a simple arity-n Boolean conjunction from only its basic components (i.e. only the *binary* Boolean AND operator and the variables necessary for the problem). In realistic applications, the required set of program components is not necessarily known in advance, and thus GP systems typically have access to a wider range of components than is strictly necessary. Hence an important question that we address in this paper is whether the GP system is able to cope with a function set containing elements which should be avoided to express the target function concisely or avoided altogether. Ideally, the system should be equipped with at least a complete set of operators, from which any Boolean function could be constructed.

In this paper, we make a considerable step forward in this direction by analysing the behaviour and performance of RLS-GP for evolving an unknown Boolean function. More precisely, the target functions we consider are either AND_n, the conjunction of n variables or OR_n, the disjunction of n variables. The GP system has access to both the binary conjunction (i.e. AND) and disjunction operators (i.e. OR). Using AND_n or OR_n as the target function simplifies our understanding of the quality of candidate solutions that mix conjunction and disjunction operators. Furthermore, we also consider a scenario where the terminal set contains all the n variables in both positive and negated forms. Thus, such a GP system is *complete* for the Boolean domain, as with its function and terminal sets it may express any possible Boolean function of n variables.

These more complex problem settings induce us to introduce more sophisticated features into the RLS-GP system than those necessary to evolve conjunctions using the AND operator alone, thus making the GP system more similar to the ones used in realistic applications. Since the presence of disjunctions in the current solution may reduce the effectiveness of the mutation operator for producing programs with better behaviour, we introduce a limit on the size of the syntax tree. This allows us to avoid issues due to bloat (a common problem for GP systems, where the size of the solution tends to increase without a corresponding increase in solution quality (Koza, 1992; Poli et al., 2008)).

With the limit on the tree size in place, our theoretical analysis reveals that the HVL-Prime mutation operator used in previous work (Durrett et al., 2011; Lissovoi and Oliveto, 2019), which either inserts, substitutes or deletes one node of the tree, may get stuck in local optima. Hence, RLS-GP with the traditional HVL-Prime operator has infinite expected runtime. To avoid this issue, we introduce a mutation mechanism which is more similar to the most commonly used subtree mutation (Koza, 1992; Poli et al., 2008). Specifically, it allows the deletion operator to remove entire subtrees in one operation, rather than limiting it to only a single leaf and its immediate parent.

We first show that RLS-GP with the above modifications is able to cope efficiently with the extended function set and the positive literals in the terminal set if it uses the complete truth table to evaluate the program quality and rejects any tree with more than $\ell = (1 + c)n$ (where $c > 0$ is a constant) leaf nodes. In particular, we prove that it evolves the exact target functions in $O(\ell n \log^2 n)$ iterations in expectation. While using the complete truth table to evaluate the program quality requires exponential time in the number of variables, we consider this setting for two main reasons. First, this setting represents the best-case model of the GP system's behaviour (i.e. a system unable to find the optimal solution when given access to a reliable fitness function is unlikely to be able to perform well with a noisy one). Second, the deterministic fitness values somewhat simplify the behaviour of the algorithm and hence our analysis. Note that if the negated literals are also included in the terminal set, then it has been proven that the standard RLS-GP cannot evolve AND_n in polynomial time with overwhelming probability even if only the AND operator is used (Mambrini and Oliveto, 2016). We conjecture that the same holds for the modified RLS-GP with extended function and terminal sets, and provide experimental evidence that this is the case.

On the other hand, we provide a positive general result for the more realistic scenario where training sets of polynomial size are sampled in each iteration uniformly at random from the complete truth table. In practice some information about the function class to be evolved may be used to decide which inputs to use in the training set. For instance, if the target function was known to be the conjunction of n variables, then a compact training set of linear size would suffice to evolve the exact solution efficiently (Lissovoi and Oliveto, 2019). However, we assume that the target function is an unknown arbitrary function composed of conjunctions and disjunctions of n variables. Our aim is to estimate the quality of the solution produced by RLS-GP in this setting.

We show that with probability $1 - O(\log^2(n)/n)$ RLS-GP is able to construct and return a conjunction (or a

---

Algorithm 1 The RLS-GP algorithm with a tree size limit ℓ.

1: Initialise an empty tree X
2: **for** $t \leftarrow 1, 2, \ldots$ **do**
3:      $X' \leftarrow \text{HVL-Prime}(X)$
4:      **if** $\text{LeafCount}(X') \leq \ell$ **and** $f(X') \leq f(X)$ **then**
5:          $X \leftarrow X'$

---

disjunction) with an arbitrarily small polynomial generalisation error in a logarithmic number of iterations even if the negated literals are present in the terminal set.

To achieve our results, we introduce a super-multiplicative drift theorem that makes use of a stronger drift than the linear one required by the traditional multiplicative drift theorem (Doerr et al., 2012b). This new contribution to the portfolio of methodologies for the analysis of randomised search heuristics (Lehre and Oliveto, 2018; Lengler, 2020) allows for the achievement of drastically smaller bounds on the expected runtime in the presence of a strong multiplicative drift.

We complement our theoretical results with an empirical investigation that confirms our theoretical intuition that leaf-only deletion may get stuck on local optima if a limit on the tree size is imposed for bloat control reasons. Additionally, the experiments indicate that while the algorithm would evolve the solution more quickly without a limit on the tree size, the size limit reduces the amount of expected undesired binary disjunction operators in the returned program.

A preliminary version of this work has previously been published at the Genetic and Evolutionary Computation Conference (Doerr et al., 2019). In this version, we additionally show that the considered GP system without further modifications can also efficiently evolve disjunctions, and provide the more general result that the algorithm is also efficient when using an ideal function and terminal set which allows for any Boolean function of up to n distinct variables to be represented. We have also extended the experimental analyses to complement the theory in this more general setting.

2 Preliminaries

In this work, we will analyse the performance of the simple RLS-GP algorithm on the AND_n and OR_n problems: the former directly, and the latter by noting that equivalent results can be derived by observing a symmetry between the search spaces of the two problems. The AND_n problem concerns the evolution of a conjunction of all n input variables while using $F = \{\text{AND}, \text{OR}\}$ binary functions and $L = \{x_1, \ldots, x_n\}$ input variables. When the program quality is evaluated using the complete truth table, the fitness function $f(X)$ counts the number of truth-value assignments (or *inputs*) on which the output of the candidate solution X differs from that of the target function $\hat{h}(\mathbf{x})$ (i.e., $\text{AND}_n(\mathbf{x}) = x_1 \wedge \ldots \wedge x_n$ and $\text{OR}_n(\mathbf{x}) = x_1 \vee \ldots \vee x_n$ for the AND_n and OR_n problems respectively). As observed by Lissovoi and Oliveto (2019), a conjunction of a distinct variables differs from AND_n on $2^{n-a} - 1$ truth-value assignments.

We will analyse the performance of the RLS-GP algorithm, which repeatedly chooses the best between its current solution and an offspring generated by applying a tree mutation operator. In addition to considering the classic HVL-Prime mutation operator, which with equal probability inserts, deletes, or substitutes a leaf node in the current solution (Durrett et al., 2011), we also propose a slightly modified version (*HVL-Prime with subtree deletion*) that is able to delete arbitrary subtrees in a single mutation.

We observe that the presence of disjunctions in the current solution may lead to bloat issues. Each OR increases the minimum number of leaf nodes required to represent the exact conjunction, and diminishes the effect of insertions beneath it on the overall program semantics. Additionally, since the classic HVL-Prime operator performs deletions by removing a leaf node and its immediate parent, it may be difficult for it to remove disjunctions placed high up in the tree. To counteract this, we add a simple bloat control mechanism to RLS-GP, making it reject trees which contain more than ℓ leaf nodes, as described in Algorithm 1.

With the tree size limit ℓ in place, applying the original HVL-Prime mutation operator Durrett et al. (2011) may cause RLS-GP to get stuck on a local optimum.

Theorem 1. *The expected optimisation time of RLS-GP with leaf-only deletion and substitution sub-operations of HVL-Prime, and any $\ell > 0$ on AND_n or OR_n with $F = \{AND, OR\}$ is infinite.*

Proof. RLS-GP may construct trees which contain ℓ leaf nodes and cannot be further improved by local mutations.

For AND_n, consider a tree constructed by initially creating a disjunction of $\ell/2$ x_1 leaf nodes, and then transforming each x_1 leaf into an $x_1 \wedge x_2$ subtree. No leaf node in the final tree can be deleted or substituted without decreasing fitness, and no insertion will be accepted due to the tree size limit, rendering RLS-GP unable

---

Algorithm 2 HVL-Prime with subtree deletion on tree X.

1: **Inputs:** a tree X, set of available literals L, set of available binary functions F.
2: Choose $op \in \{\text{INS}, \text{DEL}, \text{SUB}\}$, $l \in L$, $f \in F$ uniformly at random
3: **if** X is an empty tree **then**
4:      Set l to be the root of X.
5: **else if** $op = \text{INS}$ **then**
6:      Choose a node $x \in X$ uniformly at random
7:      Replace x with f, setting the children of f to be x and l, order chosen u.a.r.
8: **else if** $op = \text{DEL}$ **then**   ▷ modified (subtree) deletion
9:      Choose a node $x \in X$ uniformly at random
10:      Replace x's parent in X with x's sibling in X
11: **else if** $op = \text{SUB}$ **then**
12:      Choose a leaf node $x \in X$ uniformly at random
13:      Replace x with l.
14: **return** the modified tree X

---

to reach the optimum. As this tree can be constructed with non-zero probability, the expected time to construct the optimal solution is infinite by the law of total expectation.

For OR_n, an equivalent issue arises when an initial conjunction of $\ell/2$ x_1 leaf nodes has each leaf transformed into an $x_1 \vee x_2$ subtree. □

To avoid this issue, we modify the deletion operation of HVL-Prime to allow deletion of subtrees as described in Algorithm 2. The only difference with the original HVL-Prime operator is that in line 9 only choosing a leaf node for deletion was allowed while now any node may be chosen for deletion.

We use the term *sampled error* to refer to the fitness value of a particular solution in a particular iteration, and *generalisation error* to refer to the probability that a particular solution is wrong on an input chosen uniformly at random from the set of all 2^n possible inputs. When the program quality is evaluated using the complete truth table, the sampled error of a solution is always exactly 2^n times its generalisation error. When the complete truth table is used, the goal of the GP system is to construct a solution that is semantically equivalent to the target function, i.e. achieve a sampled (and generalisation) error of 0.

As it is computationally infeasible to evaluate all 2^n possible inputs for large values of n, we also analyse the behaviour of RLS-GP when evaluating the solution quality based on $s \in \text{poly}(n)$ inputs chosen uniformly at random from the set of all possible inputs. A fresh set of s inputs is chosen in each iteration, and $f(X)$, or the *sampled error*, then refers to the number of inputs, among the chosen s, on which X differs from the target function. The sampled error is thus a random variable, and its expectation is exactly s times the generalisation error of the solution. We bound the probability that the sampled error deviates from its expectation in Lemma 1 below. When a polynomially sized training set is used to evaluate the program quality, the goal of the GP system is to construct a solution with a low generalisation error. On AND_n, and most other non-trivial problems, we do not expect the GP systems to reach a generalisation error of 0 while s remains polynomial with respect to the problem size, unless the problem's fitness landscape is well understood and a problem-specific training set is used (Lissovoi and Oliveto, 2019). We assume that this is not the case, and that the aim is to find a solution that has an arbitrarily small polynomial generalisation error. Note that we use the following notation throughout the paper: $\mathbb{N} := \{0, 1, 2, \ldots\}$, $\lg(n)$ and $\ln(n)$ denoting the base 2 and the natural logarithms of n, while $\log n$ is used in asymptotic bounds.

Lemma 1. *Let $s \in \text{poly}(n)$ be the number of inputs sampled by the GP system, G be the generalisation error of a solution, and X be the random variable that denotes the sampled error of that solution. Then, for any c that is at least a positive constant,*

$$|Gs - X| \leq \max\{c \lg n, Gs\}$$

with probability at least $1 - n^{-\Omega(c)}$.

Proof. X is a sum of s Bernoulli variables, each with a probability G of assuming the value 1 (and 0 otherwise), and hence $E[X] = Gs$. As both X and Gs are non-negative, $Gs - X \leq Gs$, and we focus solely on the case where X significantly exceeds its expectation, the probability of which can be bounded by applying a Chernoff bound.

Suppose that $E[X] \geq (c/2) \lg n$; then, $\Pr[X \geq (1 + 1)E[X]] \leq e^{-E[X]/3} \leq n^{-\Omega(c)}$; and hence $|Gs - X| < Gs$, with probability at least $1 - n^{-\Omega(c)}$. Otherwise, we upper bound $E[X] \leq \mu^+ = (c/2) \lg n$, and apply a

Chernoff bound using μ^+ Doerr (2020, Theorem 1.10.21), obtaining $\Pr[X \geq (1+1)\mu^+] \leq e^{-\mu^+/3} = n^{-\Omega(c)}$; and hence $|Gs - X| \leq X \leq c\lg n$ with probability at least $1 - n^{-\Omega(c)}$. $\qquad\square$

3 Complete truth table

In this section, we will present a runtime analysis of the RLS-GP algorithm with subtree deletion (i.e. Algorithm 1 using Algorithm 2 as the mutation operator) on the AND_n problem, using the complete truth table to evaluate the solution quality, i.e., executing each constructed program on all 2^n possible inputs.

Theorem 2. *The expected runtime of RLS-GP with $F = \{AND, OR\}$, $L = \{x_1, \ldots, x_n\}$, and $\ell \geq n$ on AND_n or OR_n is $E[T] = \Omega(n \log n)$.*

Proof. No tree which does not contain all n distinct variables can be equivalent to the AND_n or OR_n functions. By a standard coupon collector argument, $\Omega(n \log n)$ insertion or substitution operations are required to insert all n distinct variables into the tree. $\qquad\square$

The following drift theorem requires that the expected progress when at distance d from the target is of order $\Omega(d \log d)$. This assumption is slightly stronger than the linear (i.e. $\Omega(d)$) progress assumed in the multiplicative drift theorem. Despite this apparently small difference, the resulting bounds for the expected time to reach the target differ drastically. For an initial distance of d_0, they are, roughly speaking, $O(\log d_0)$ for the multiplicative drift situation and $O(\log \log d_0)$ for our super-multiplicative drift.

Theorem 3 (Super-multiplicative drift theorem). *Let $\gamma > 1$ and $\delta > 0$. Let $X_0, X_1, \ldots$ be random variables taking values in $\Omega = \{0\} \cup [1, \infty)$. Assume that for all $t \in \mathbb{N}$ and all $x \in \Omega \setminus \{0\}$ such that $\Pr[X_t = x] > 0$ we have*

$$E[X_t - X_{t+1} \mid X_t = x] \geq (\log_\gamma(x) + 1)\delta x. \tag{1}$$

Then the first hitting time $T = \min\{t \in \mathbb{N} \mid X_t = 0\}$ of zero satisfies

$$E[T \mid X_0] \leq \frac{3}{\delta} + \frac{2(2 + \log_2 \log_\gamma \max\{\gamma, X_0\}) \ln \gamma}{\delta}.$$

Proof. For all $k \in \mathbb{N}_{\geq 1}$, let $T_k := \min\{t \in \mathbb{N} \mid X_t < \gamma^{2^{k-1}}\}$. We first show that

$$E[T_k - T_{k+1}] \leq \frac{1 + 2^k \ln \gamma}{(2^{k-1} + 1)\delta}$$

holds for all $k \geq 1$. To this aim, we regard the process Y_t defined for all $t \in \mathbb{N}$ by $Y_t = X_t$ if $t \leq T_k - 1$ and $Y_t = 0$ otherwise. By definition, $T_k^Y := \min\{t \in \mathbb{N} \mid Y_t < \gamma^{2^{k-1}}\}$ satisfies $T_k^Y = T_k$. We show that the process (Y_t) satisfies the multiplicative drift condition,

$$E[Y_t - Y_{t+1} \mid Y_t] \geq (2^{k-1} + 1)\delta Y_t.$$

If $t \geq T_k$, then both $Y_t = 0$ and $Y_{t+1} = 0$. Consequently, the multiplicative drift condition is trivially satisfied. In the more interesting case that $t < T_k$, we have $Y_t \geq \gamma^{2^{k-1}}$ and $Y_t = X_t$. From this, $Y_{t+1} \leq X_{t+1}$, and (1), we conclude

$$E[Y_t - Y_{t+1} \mid Y_t] \geq E[X_t - X_{t+1} \mid X_t] \geq (\log_\gamma(X_t) + 1)\delta X_t \geq (2^{k-1} + 1)Y_t,$$

again showing the multiplicative drift condition.

Let $T^Y := \min\{t \in \mathbb{N} \mid Y_t = 0\}$. Since $T^Y = T_k^Y = T_k$ and since $Y_t \leq \gamma^{2^k}$ for all $t \geq T_{k+1}$, the multiplicative drift theorem (Doerr et al., 2012b) yields $E[T_k - T_{k+1}] = E[T^Y - T_{k+1}^Y] \leq \frac{1 + \ln \gamma^{2^k}}{(2^{k-1} + 1)\delta} = \frac{1 + 2^k \ln \gamma}{(2^{k-1} + 1)\delta}$.

By a simple application of the multiplicative drift theorem, we also observe that $E[T - T_1] \leq \frac{1 + \ln \gamma}{\delta}$.

In the following, we condition on the initial value X_0. Assume that $X_0 \in [\gamma^{2^{k-1}}, \gamma^{2^k})$ for some $k \in \mathbb{N}_{\geq 1}$. Then $T_{k+1} = 0$ and thus $T = \sum_{i=1}^{k}(T_i - T_{i+1}) + (T - T_1)$. We compute

$$E[T] = \sum_{i=1}^{k} E[T_i - T_{i+1}] + E[T - T_1] \leq \sum_{i=1}^{k} \frac{1 + 2^i \ln \gamma}{(2^{i-1} + 1)\delta} + \frac{1 + \ln \gamma}{\delta}$$

$$\leq \frac{3}{\delta} + \frac{2(k+1)\ln \gamma}{\delta} \leq \frac{3}{\delta} + \frac{2(2 + \log_2 \log_\gamma X_0)\ln \gamma}{\delta}.$$

For $X_0 < \gamma$, we have in an analogous way $E[T] \leq \frac{1 + \ln(X_0)}{\delta} \leq \frac{1 + \ln \gamma}{\delta}$. This proves the claim. $\qquad\square$

The proof of Theorem 3 estimates the super-multiplicative drift by piece-wise multiplicative drifts. We preferred this proof method because of its simplicity and because it could, by using the multiplicative drift theorem with tail-bounds (Doerr and Goldberg, 2013), also lead to tail-bounds for super-multiplicative drift as well[1]. An alternative approach which would improve the time bound by a constant factor (again a feature we are not interested in here) would be to use variable drift (Mitavskiy et al., 2009; Johannsen, 2010).

We use the super-multiplicative drift theorem to prove our upper bound on the expected runtime of RLS-GP when using the complete truth table as the training set. We initially focus on the AND_n problem, and will later show that because of the symmetry in the OR_n and the AND_n problems, equivalent results also hold for OR_n. We begin by bounding the time spent in iterations in which the tree is not full, i.e. it has not reached the size limit of having ℓ leaf nodes.

Lemma 2. *Consider a run of RLS-GP on* AND_n*, with* $F = \{AND, OR\}$*,* $L = \{x_1, \ldots, x_n\}$*, and a tree size limit of* $\ell \geq n$*. Let* T *be the number of iterations before the optimum is found, and* $T_0 \leq T$ *be the number of these iterations in which the parent individual is not a full tree. Then,* $E[T_0] = O(\ell\, n \log^2 n)$*.*

Proof. To bound $E[T_0]$, we will apply Theorem 3 using the solution fitness as the potential function, and considering only the iterations in which the tree is not full. While the tree *is* full, we instead rely on the elitism of the RLS-GP algorithm to not accept mutations which increase the potential function value (i.e., offspring with a worse fitness value). Thus, the T_0 iterations in which the tree is not full need not be contiguous.

In an iteration starting with a tree containing less than ℓ leaf nodes, it is possible to insert a new leaf node x_i with an AND parent anchored at the root of the tree. We call such an operation a root-and. The probability that in one iteration a root-and with a fixed variable x_i is performed, is at least $\frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2\ell} \cdot \frac{1}{n} = \frac{1}{12\ell n}$.

We compute the expected fitness gain caused by such modifications. Because the fitness never worsens, it suffices to regard certain operations that improve the fitness. Recall further that the fitness is just the number of assignments to the variables $x_1, \ldots, x_n$ such that the tree evaluates differently from AND_n.

Let $x_1, \ldots, x_n$ be such an assignment. This implies that not all x_i are true, because any tree generated by RLS-GP evaluates correctly to true for the all-true assignment. Assume that exactly $k \geq 1$ of the variables $x_1, \ldots, x_n$ are false, but that our tree solution evaluates to true. Then there are exactly k variables such that a root-and with one of them would make the program evaluate to false on this assignment (and thus improve the fitness, since false is the correct output). The probability for such a mutation is at least $\frac{k}{12\ell n}$.

For any $1 \leq i \leq n$, there are exactly $\binom{n}{i}$ inputs where exactly i variables are set to false, and exactly $\sum_{i=1}^{k-1} \binom{n}{i}$ inputs where less than k variables are set to false. Thus, if the fitness of the current solution is at least $M_k = 2\sum_{i=1}^{k-1} \binom{n}{i}$, at least half of the inputs contributing to the fitness have at least k variables set to false.

By only regarding the progress caused by these, we have, for $x \geq M_k$,

$$E\left[f(X^t) - f(X^{t+1}) \mid f(X^t) = x\right] \geq \frac{1}{12\ell}\frac{k}{n}\,x. \tag{2}$$

Since for n sufficiently large we have $M_k \leq 2n^{k-1}$ for all $k \in [1..n]$. This implies that for all $x \in [1..2^n]$ and all $t \in \mathbb{N}$, we have

$$E\left[f(X^t) - f(X^{t+1}) \mid f(X^t) = x\right] \geq \tfrac{1}{12\ell n}\left(\lfloor \log_n(x/2)\rfloor + 1\right)x$$
$$\geq \tfrac{1}{36\ell n}\left(\log_n(x) + 1\right)x,$$

where the last estimate uses $n \geq 2$. Hence, Theorem 3 with $\gamma = n$ and $\delta = 1/36\ell n$ gives

$$E[T] \leq 36\ell n(3 + 2(2 + \log_2 \log_n 2^n))\ln n = O(\ell n \log^2 n).$$

$\square$

To prove following theorem, we will show that with high probability, the parent solution contains fewer than ℓ leaf nodes in at least a constant fraction of any $t \in \Omega(\ell\, n \log^2 n)$ iterations. Intuitively, this means that the conditions of Lemma 2 apply often enough to not affect the asymptotic expected runtime (i.e. $E[T] = O(E[T_0])$).

Theorem 4. *Consider a run of RLS-GP on* AND_n*, using* $F = \{AND, OR\}$*,* $L = \{x_1, \ldots, x_n\}$*, and a tree size limit of* $\ell = (1 + c)n$*. Let* T *be the number of iterations before the optimum is found. If* $c = \Theta(1)$*, then* $E[T] = O(\ell\, n \log^2 n)$*.*

---

[1] For this, we note that the multiplicative drift theorem with tail bounds (Doerr and Goldberg, 2013) shows that the phase lengths $T_k - T_{k-1}$ and $T - T_1$ are stochastically dominated (see, e.g., Doerr (2019)) by their expectation plus a geometrically distributed random variable. Consequently, the deviation of T above its expectation is stochastically dominated by a sum of independent geometrically distributed random variables and thus can be bounded by tail bounds for such sums (Doerr, 2020, Section 1.10.4). We omit further details since in this work we do not need such tail bounds.

Proof. Let $T' = c^* \ell n \log^2 n$, for some constant $c^* > 0$, be an upper bound on the expected number of iterations $E[T_0]$ in which the tree is not full before the optimum solution is found per Lemma 2. By an application of Markov's inequality, the probability that the optimum is found in at most $2T'$ such iterations is at least $1/2$. We will show that if $\ell = (1+c)n$, for any constant $c > 0$, $2T'$ such iterations occur in $(2+c')T'$ iterations with high probability, where $c' > 0$ is constant with respect to n. The theorem statement then follows from a simple waiting time argument: during each period of $(2+c')T'$ iterations, the optimum is found with probability at least $1/2 \cdot (1-o(1)) = \Omega(1)$, so the expected number of such periods before the optimum is found is at most $O(1)$, and thus the expected runtime is at most $O(T') = O(\ell n \log^2 n)$ iterations.

We will now show that during any $N \in \Omega(\ell n \log^2 n)$ iterations, with high probability and for some constant $c'' > 0$, deletions of at least $c'' N$ leaf nodes in total will be accepted. As each iteration can at most increase the number of leaf nodes in the tree by 1, there will with high probability be at least $c'' N$ iterations in which the tree is not full among any $(1+c'')N$ iterations. As $T' \in \Omega(\ell n \log^2 n)$, $2T'$ iterations in which the tree is not full will with high probability occur in $(2+c')T'$ iterations where $c' = 2/c'' = \Omega(1)$.

Consider a tree X with exactly ℓ leaf nodes. Let $L_A(X)$ be a set of leaf nodes connected to the root of X via only AND nodes, and call *essential* all the leaf nodes in this set that contain a variable which only appears on nodes in this set exactly once. If X is non-optimal, at most $n-1$ leaf nodes in X are essential, and at least $\ell - (n-1)$ leaf nodes are non-essential. All non-essential nodes are either directly deletable (in the case of redundant copies of variables in $L_A(X)$), or indirectly deletable (by deleting a branch at any of their OR ancestors).

Every non-essential leaf node can thus be deleted by performing an HVL-Prime deletion sub-operation on at least one node in the tree. For some non-essential leaf nodes, a larger subtree may need to be deleted to remove the leaf without adversely impacting fitness. The longer waiting time for such subtree deletions (requiring that the root of the subtree be chosen for deletion rather than one of the many leaf nodes in the subtree) is balanced by the increased number of leaf nodes deleted as part of the mutation. We note that the tree contains $2\ell - 1$ nodes, and thus for $\ell \geq (1+c)n$ and any $c > 0$, an HVL-Prime mutation in expectation reduces the number of leaf nodes in the tree by at least

$$\frac{1}{3} \frac{\ell - (n-1)}{2\ell - 1} \geq \frac{\ell - n}{6\ell} \geq \frac{c}{6+6c} \geq \delta \in \Omega(1),$$

where $\delta > 0$ is a positive constant, as $c \in \Omega(1)$.

Let $X_1, \ldots, X_N$ be the number of leaf nodes deleted in an accepted mutation during each iteration performed while the tree is full, and $X = \sum_{i=1}^{N} X_i$. Furthermore, define a sequence $Z_0, \ldots, Z_N$, where $Z_0 := 0$ and $Z_i := Z_{i-1} + X_i - \delta$; clearly, $Z_N - Z_0 = Z_N = X - \delta N$. We will show that $Z_N > -\delta N/2$ (and therefore $X > \delta N/2 \in \Omega(N)$) holds with high probability.

As $E[Z_i \mid Z_1, \ldots Z_{i-1}] = Z_{i-1} + E[X_i \mid Z_1, \ldots Z_{i-1}] - \delta \geq Z_{i-1}$, the sequence $Z_0, \ldots, Z_N$ is a sub-martingale, and $c_i := |Z_i - Z_{i-1}| \leq \ell$. Hence, by applying the Azuma-Hoeffding inequality for $N \in \Omega(\ell n \log^2 n)$ and $t = \delta N/2$,

$$\Pr[Z_N - Z_0 \leq -t] \leq \exp\left(\frac{-t^2}{2\sum_{i=1}^{N} c_i^2}\right) \leq \exp\left(\frac{-\delta^2 N}{8\ell^2}\right) \leq n^{-\Omega(\log n)}$$

as $N/\ell^2 = \Omega(n\ell \log^2 n/\ell^2) = \Omega(\log^2 n)$ for $\ell = (1+c)n$ where c is a constant.

Thus, there exists a constant $c'' > 0$ such that over the course of $N \in \Omega(n\ell \log^2 n)$ iterations where the tree is full, deletions of at least $\delta N/2 = c'' N$ leaf nodes are accepted with high probability, and hence over the course of $(2/c'')N$ iterations, at least $2N$ iterations occur while the tree is not full with high probability. Setting $N = T' = c^* n\ell \log^2 n$ iterations per Lemma 2 completes the proof: among $\Theta(T')$ iterations, at least $\Omega(T')$ will take place while the tree is not full, allowing the application of the Markov inequality and waiting time arguments to produce the bound on the expected runtime. $\square$

Additionally, we prove that the RLS-GP algorithm using the same function and terminal sets is able to evolve disjunctions of n variables efficiently when using the complete truth table to evaluate the solution fitness. Theorem 5 formalises this result, and is proven by the same methods as Theorem 4, taking advantage of the symmetry between the search spaces of the AND_n and OR_n problems.

Theorem 5. *Consider a run of RLS-GP on OR_n, using $F = \{AND, OR\}$, $L = \{x_1, \ldots, x_n\}$, and a tree size limit of $\ell = (1+c)n$. Let T be the number of iterations before the optimum is found. If $c = \Theta(1)$, then $E[T] = O(\ell n \log^2 n)$.*

Proof. We note that there is a symmetry between the output vectors of the AND_n and OR_n target functions: while AND_n returns true only when all n input variables are true, OR_n returns false only when all n input variables are false. Similarly, a disjunction of a distinct variables is wrong on $2^{n-a} - 1$ inputs on OR_n.

Lemma 2 carries over to the OR_n problem by considering the time taken to find the optimal solution through insertions of *disjunctions* at the top of the tree, observing that for any $1 \leq i \leq n$, there are exactly $\binom{n}{i}$ inputs where exactly i variables are set to *true*, and exactly $\sum_{i=1}^{k-1} \binom{n}{i}$ inputs where less than k variables are set to *true*. Thus, if the fitness of the current solution is at least $M_k = 2\sum_{i=1}^{k-1} \binom{n}{i}$, at least half of the inputs contributing to the fitness have at least k variables set to *true*, allowing the error to be at least halved by inserting one of these k variables.

The proof of Theorem 4 then carries over to the OR_n problem by adjusting the definition of essential leaf nodes: let $L_A(X)$ be a set of leaf nodes connected to the root of X via only *OR* nodes, and call *essential* all the leaf nodes in this set that contain a variable which only appears on nodes in this set exactly once. A full tree can then be shrunk by removing non-essential nodes, either directly (in the case of redundant copies of variables in $L_A(X)$) or by deleting a branch at any of their AND ancestors.

Thus, the runtime bound derived in the proof of Theorem 4 also holds for the OR_n problem. $\qquad\square$

If the terminal set is extended by adding negations of all positive input variables, RLS-GP with $F = \{AND, OR\}$ is able to represent any Boolean function (by having a tree expressing its disjunctive or conjunctive normal form, where any negations are pushed all the way down to individual variables). For the AND_n problem with $F = \{AND\}$, Lissovoi and Oliveto (2019) have shown that such a terminal set extension renders RLS-GP inefficient when using the complete training set. In that setting, any solution containing a contradiction (e.g., $x_1 \wedge \overline{x}_1$) will have the next-to-optimal fitness (being wrong on only the all-true input). This deprives RLS-GP of a fitness signal until an optimal solution is constructed by random chance inserting all positive terminals into the tree and removing copies of all negative terminals from the tree. Since all mutations that do not remove the last-remaining contradiction are accepted, this requires at least exponential time in expectation.

We conjecture that a similar result would hold for RLS-GP with $F = \{AND, OR\}$, and subtree deletion on both the AND_n and OR_n problems, as trees containing a contradiction on AND_n, or a tautology on OR_n, can be constructed, and would only be wrong on a single input. However, the negative drift analysis proof technique used in Lissovoi and Oliveto (2019) cannot be easily applied with the subtree mutation operator as large jumps in the search space may occur with high probability. Thus we leave this conjecture for future work, and focus in the next section on the positive result for the realistic scenarios where training sets of polynomial size are used.

4 Polynomially sized training sets

While the previous section provides polynomial bounds on the number of iterations required to evolve a conjunction or a disjunction of all n variables, calculating the solution quality by comparing the programs' output to that of the target function on all of the 2^n possible inputs in each iteration requires exponential computational effort. Thus, it is only computationally feasible for evolving Boolean functions of modest size.

In this section, we consider the behaviour of the RLS-GP algorithm using only a polynomial computational effort in each iteration. To this end, the solution quality is compared by evaluating the output of the parent and offspring solutions and the target function on only a polynomial number of inputs (the "training set") sampled uniformly at random from the set of all possible inputs in each iteration. This setting was previously considered by Lissovoi and Oliveto (2019), where it was shown that RLS-GP and (1+1) GP using $F = \{AND\}$ are able to construct a solution with $O(\log n)$ distinct variables which fits a random polynomially large training set in expected $O(\log n)$ time.

For our main theoretical result below, we opt to have RLS-GP terminate and return a solution once the sampled error on the training set is below a logarithmic acceptance threshold. This effectively prevents RLS-GP from entering a region of the search space where the mechanism it uses to evaluate the program quality is overly noisy. This slightly decreases the expected solution quality, but still allows to guarantee arbitrarily small generalisation errors.

Theorem 6. *For any constant $c > 0$, consider an instance of the RLS-GP algorithm with $F = \{AND, OR\}$, $L = \{x_1, \ldots, x_n\}$, $\ell \geq n$, using a training set of $s = n^c \lg^2 n$ rows sampled uniformly at random from the complete truth table in each iteration to evaluate the solution quality, and terminating when the sampled error of the solution is at most $c' \lg n$, where c' is an appropriately large constant. For both the AND_n and OR_n problems, with probability at least $1 - O(\log^2(n)/n)$, the algorithm will terminate within $O(\log n)$ iterations, and return a solution with a generalisation error of at most n^{-c}.*

To prove this theorem, we will show that RLS-GP is able to create a tree that contains no more than one copy of each variable, no undesired functions (i.e., OR and AND on the AND_n and OR_n problems respectively), and enough distinct variables to sample an error below the acceptance threshold within $O(\log n)$ iterations with

probability at least $1 - O(\log^2(n)/n)$. Additionally, we will show that with high probability the GP system will not terminate early (i.e. it will not return a solution with a generalisation error greater than n^{-c} for an arbitrary constant c).

Lemma 3. *If, in the setting of Theorem 6, RLS-GP never accepts solutions containing undesired function nodes (i.e., OR and AND on the AND_n and OR_n problems respectively) or multiple copies of any variable, and never accepts solutions with a worse generalisation error than their ancestors, it will within $O(\log n)$ iterations reach a solution with a sampled error below $c' \lg n$, where $c' > 0$ is an appropriate constant, with probability at least $1 - O(1/n)$.*

Proof. To ensure that an error below $c' \lg n$ is sampled, we consider the time required to construct a solution with an expected sampling error of at most $(c'/4) \lg n$. Such a sampling error can be achieved by a generalisation error of at most $((c'/4) \lg n)/(n^c \lg^2 n) = (c'/4)n^{-c}/\lg n \geq n^{-(c+1)}$ (for a sufficiently large n), i.e. a conjunction (on the AND_n problem, or a disjunction on the OR_n problem) of $(c + 1) \lg n$ variables or more.

The time required to construct such a solution under the lemma's conditions can be bounded by lower-bounding the probability of inserting a new variable connected to the tree using an appropriate function node (i.e., AND on AND_n and OR on OR_n), and using a Chernoff bound to show that a sufficient number of such insertions occur within a particular number of iterations (as the number of distinct variables in the current solution is never reduced by the lemma's conditions). Specifically, suppose that the current solution contains $i < n/2$ distinct variables and no undesired function nodes, and let X_i be the event that a mutation inserts a new variable and connects it to the tree using an appropriate function node, and is accepted. We bound $\Pr[X_i] \geq (1/3)(1/2)(n - i)/n \geq \delta$, i.e. $\delta \geq 1/12$ for $i < n/2$. The probability that at least $(c + 1) \lg n$ such mutations are accepted within $(c''/\delta)(c + 1) \lg n = O(\log n)$ iterations is then, by applying a multiplicative Chernoff bound, at least $1 - e^{-\Omega(c'' \log n)} = 1 - n^{-\Omega(c'')}$. Thus, when c'' is a sufficiently large constant, this probability is at least $1 - O(1/n)$.

We bound the probability that a solution with a low-enough expected sampled error does not meet the acceptance threshold by applying Lemma 1. Once a solution with an expected sampled error of at most $(c'/4) \lg n$ is constructed, the probability that its sampled error exceeds the acceptance threshold is at most $n^{-\Omega(c')}$. Thus, when c' is picked appropriately, the solution is accepted immediately with probability at least $1 - O(1/n)$.

By combining the failure probabilities using a union bound, we conclude that RLS-GP under the conditions of the lemma and with an appropriately-chosen constant c', is able to construct a solution with an acceptable sampled error within $O(\log n)$ iterations with probability at least $1 - O(1/n)$. □

We will now use this bound on the runtime of RLS-GP to show that it is likely to avoid all of the potential pitfalls preventing the application of Lemma 3.

Lemma 4. *In the setting of Theorem 6, with probability at least $1 - O(\log^2(n)/n)$, during the first $O(\log n)$ iterations and while the expected sampled error of its current solution remains above $(c'/4) \lg n$, RLS-GP is able to avoid accepting mutations which: (1) insert copies of a variable already present in the current solution, (2) insert undesired function nodes (i.e., OR and AND on the AND_n and OR_n problems respectively), or (3) increase the generalisation error of the current solution.*

Proof. For claim (1), we note that within the first $O(\log n)$ iterations, the tree will contain at most $O(\log n)$ distinct variables (as each iteration of RLS-GP is only able to insert one additional variable). Thus, the probability that a mutation operation adds a variable which is already present in the solution (using either the insertion or substitution sub-operation of HVL-Prime) is at most $O(\log n/n)$. By a union bound, this does not occur during the first $O(\log n)$ iterations with probability at least $1 - O(\log^2(n)/n)$.

For claim (2), we note that there are two main ways an undesired function node can be introduced into the solution by an insertion operation. First, the function can be semantically neutral, e.g. on the AND_n problem, if the ancestor contains only ANDs and unique variables, a leaf x_i could be replaced with $x_i \lor x_i$ without affecting any of the solution's outputs. Second, the sampling process used to evaluate the solution fitness may not have sampled any inputs on which the offspring is wrong and the ancestor is correct. We will consider the two possibilities separately.

As semantically-neutral insertions of undesired function nodes require inserting a duplicate copy of a variable, claim (1) already provides the desired probability bound on these insertions not occurring within $O(\log n)$ iterations (and hence not being accepted). All other undesired function node insertions will increase the generalisation error of the solution. The magnitude of this increase depends on the number of distinct variables in the subtree displaced by the insertion, with insertions displacing only a single leaf node being the easiest to accept.

If a leaf of the ancestor solution is displaced by an insertion which adds an undesired function node and a new variable, we use the term *witness* to refer to inputs on which the constructed offspring solution produces incorrect

9

output while the parent solution produces correct output. For the AND_n problem, *witness* inputs are those where the displaced variable to false while setting the remaining variables in the offspring solution to true, while on the OR_n problem, the displaced variable is set to true while the remaining variables in the offspring solution are set to false. Witness inputs demonstrate that the offspring solution is worse than its parent; on all other inputs, the parent and offspring produce the same output. Thus, as long as the sampling procedure finds at least one witness, RLS-GP will reject the mutated solution.

Suppose the ancestor solution contains U distinct variables (and uses only the desired function nodes, i.e., AND on AND_n and OR on OR_n); it is then incorrect on $2^{n-U} - 1$ possible inputs, while there are at least $2^{n-(U+1)}$ witnesses; i.e., the probability of randomly selecting a witness is at least half that of randomly selecting an input on which the ancestor is wrong. Thus, if the expected sampled error of the ancestor solution is at least X, the expected number of witnesses in the sample is at least $X/2$. By a Chernoff bound, the probability that fewer than $(c'/16)\lg n$ witnesses are present in the sample is at most $e^{-(c'/128)\lg n} = n^{-\Omega(c')}$. By setting the constant c' appropriately, this probability can be made into $O(1/n)$. Hence, by a union bound, the probability that no insertion of an undesired function node which increases the generalisation error is accepted within $O(\log n)$ iterations while the expected sampled error of the solution remains above $(c'/4)\lg n$ is at least $1 - O(\log(n)/n)$.

For claim (3), we note that decreasing the number of distinct variables in the solution more than doubles its generalisation error. Applying a similar argument as for rejecting detrimental undesired function node insertions above (with at least X witnesses expected in a sampled training set), the probability that no mutations increasing the generalisation error are accepted during $O(\log n)$ iterations is at least $1 - O(\log(n)/n)$.

Combining the error probabilities of the three claims using a union bound yields the theorem statement. $\qquad\square$

Finally, we show that with high probability, RLS-GP does not terminate unacceptably early (i.e., by sampling an error below the acceptance threshold for a solution with a worse generalisation error than desired by Theorem 6).

Lemma 5. *In the setting of Theorem 6, with high probability, no solution with a generalisation error greater than n^{-c} has a sampled error of at most $c' \lg n$ on a set of $s \geq n^c \lg^2 n$ rows sampled random from the complete truth table, within any polynomial number of iterations.*

Proof. Recall that when sampling s rows uniformly at random from the complete truth table to evaluate the solution fitness, RLS-GP terminates and returns the current solution when the solution appears wrong on at most $c' \lg n$ of the sampled rows. As the generalisation error of a solution is also the probability that the solution is wrong on a uniformly-sampled row of the complete truth table, a solution X with a generalisation error $g(X)$ of at least n^{-c}, has an expected sampled error $E(f(X)) \geq \lg^2 n$ on $s = n^c \lg^2 n$ rows sampled uniformly at random. Applying a Chernoff bound, the probability that the sampled error Y is less than half of its expected value (which for large-enough n is above the $c' \lg n$ threshold), is super-polynomially small:

$$\Pr\left[Y \leq 1/2 \, E[Y]\right] \leq e^{-E[Y]/8} \leq n^{-\Omega(\log n)}.$$

By a union bound, RLS-GP with high probability does not return a solution with a generalisation error of at least n^{-c} within any polynomial number of iterations when sampling $s = \Omega(n^c \lg^2 n)$ rows of the complete truth table uniformly at random to evaluate the solution quality in each iteration. $\qquad\square$

Our main result is proved by combining these lemmas.

Proof of Theorem 6. By Lemma 4, Lemma 3 can be applied with probability at least $1 - O(\log^2(n)/n)$, and thus with probability at least $1 - O(\log^2(n)/n) - O(1/n)$, a solution with a sampled error meeting the acceptance threshold will be found and returned within $O(\log n)$ iterations. By Lemma 5, the generalisation error of any solution returned by RLS-GP within a polynomial number of iterations is with high probability better than the desired n^{-c}. $\qquad\square$

Performing λ runs of RLS-GP, as is often done in practice, and terminating once any instance determines that its current solution meets the acceptance threshold, will guarantee that a solution with the desired generalisation error is produced using $O(\lambda \log n)$ fitness evaluations with probability $1 - n^{-\Omega(\lambda)}$.

We now consider the GP system equipped with complete Boolean function and terminal sets i.e., the algorithm has the capacity to represent any Boolean function of up to n distinct variables. We show that when using a polynomial training set to evaluate the solution quality, extending the terminal set to include negations of input variables (thus allowing RLS-GP to represent any Boolean function) does not significantly affect the algorithm's ability to produce solutions with desired polynomially-small generalisation errors on the AND_n and OR_n problems. This result is formalised in Theorem 7, which is proven by observing that insertions of negated variables are almost as useful as insertions of positive variables on these problems.

Experiment	Deletion	F	L	Training Set	Runs	Max Iterations
RQ 1	leaf	$\{AND, OR\}$	$\{x_1, \ldots, x_n\}$	Complete	500	10,000
RQ 2	subtree	$\{AND, OR\}$	$\{x_1, \ldots, x_n\}$	Complete	500	10,000
RQ 3	subtree	$\{AND, OR\}$	$\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$	Complete	500	10,000
RQ 4	subtree	$\{AND, OR\}$	$\{x_1, \ldots, x_n\}$	Incomplete	500	10,000
RQ 5	subtree	$\{AND, OR\}$	$\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$	Incomplete	500	10,000

Table 1: The experimental setup used in each of the five carried out experiments.

Theorem 7. *For any constant $c > 0$, consider an instance of the RLS-GP algorithm with $F = \{AND, OR\}$, $L = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$, $\ell \geq n$, using a training set of $s = n^c \lg^2 n$ rows sampled uniformly at random from the complete truth table in each iteration to evaluate the solution quality, and terminating when the sampled error of the solution is at most $c' \lg n$, where c' is an appropriately large constant. For both the AND_n and OR_n problems, with probability at least $1 - O(\log^2(n)/n)$, the algorithm will terminate within $O(\log n)$ iterations, and return a solution with a generalisation error of at most n^{-c}.*

Proof. This result can be proven by following the proof of Theorem 6, supplemented by a number of observations regarding the utility of adding negated variables into the current solution.

A solution containing both the positive and negative form of some variable and only problem-appropriate function nodes (i.e., AND for AND_n and OR for OR_n; producing a contradiction or a tautology respectively) trivially achieves the desired polynomially-small generalisation error, as it is only wrong on a single input out of 2^n possible inputs. Typically, however, RLS-GP will terminate with a solution of desired quality before inserting both the positive and negative copies of some variable.

Any solution composed of problem-appropriate function nodes and a mix of positive and negative input variables, but not both the positive and negative versions of any one variable, is wrong on exactly two more inputs than a solution which contains only positive versions of the same variables. For the AND_n problem, these two inputs are the all-true input (on which the solution containing negated variables return false), and the input setting all used leaf nodes to true by assigning true to variables appearing in positive form, and false to variables appearing negated, and all remaining variables to true. Thus, the first insertion of a negated variable (which does not already appear in the tree in positive form) is only slightly worse than inserting the corresponding positive variable, while subsequent insertions of negated and positive variables are equivalent as long as no contradiction or tautology is produced. The difference is overwhelmingly unlikely to be noticed when sampling a training set of polynomial size for any polynomial number of iterations: these two inputs are not selected for any polynomially-sized training set within a polynomial number of iterations with probability overwhelmingly close to 1 by a union bound, and conditioning on this, the RLS-GP is as likely to accept/reject a mutation involving a negated literal as a positive literal of the same variable.

For the purposes of Lemmas 3 and 4, the negated variables can therefore be treated as being equivalent to their positive counterparts, with any solution which contains a logarithmic number of distinct variables in either form and only the problem-appropriate function nodes producing the desired generalisation error. Thus, with high probability, a tree which contains each variable in either positive or negative form at most once, contains no undesired function nodes, and achieves the desired generalisation error can be constructed within $O(\log n)$ iterations.

Combining these with Lemma 5 as in the proof of Theorem 6, and including the probability of not sampling a training set capable of distinguishing negations in the final union bound yields the result. □

5 Experiments

We performed experiments to complement our theoretical results. For each choice of algorithm and problem parameters, we performed 500 independent runs of the GP systems and we report the average number of iterations required to find the solution, its average size, as well as standard deviations.

We will investigate five separate research questions to expand upon the knowledge gained from the theory regarding RLS-GP for evolving conjunctions. The experimental setup for each research question is reported in Table 1.

We first examine the behaviour of the algorithms when they use the complete truth table to evaluate solution quality. Theorem 1 showed that using the standard HVL-Prime operator, which applies leaf-only deletion and substitution, can cause RLS-GP with the complete truth table to get stuck on a local optimum when a tree size limit is imposed, thus leading to infinite expected runtime. However, the theorem does not provide bounds on the probability that this event occurs.

	$\ell = n$			$\ell = n+1$		
n	B	$\overline{T}$	$\overline{S}$	B	$\overline{T}$	$\overline{S}$
4	0.008	46.3 (28.0)	4.0 (0.0)	0.002	40.9 (21.8)	4.4 (0.5)
8	0.002	151.8 (91.9)	8.0 (0.0)	0.004	113.8 (51.5)	8.6 (0.5)
12	0.016	284.1 (148.2)	12.0 (0.0)	0.002	214.3 (99.5)	12.7 (0.5)
16	0.008	469.9 (258.0)	16.0 (0.0)	0.010	345.8 (161.0)	16.8 (0.4)

	$\ell = 2n$			$\ell = \infty$		
n	B	$\overline{T}$	$\overline{S}$	B	$\overline{T}$	$\overline{S}$
4	0	42.5 (25.8)	5.1 (1.2)	0	38.9 (24.3)	5.4 (2.0)
8	0	98.8 (49.0)	11.0 (2.3)	0	95.3 (43.8)	11.2 (3.0)
12	0	170.7 (99.7)	17.1 (3.3)	0	160.1 (57.1)	17.9 (4.5)
16	0	232.5 (80.9)	23.8 (4.1)	0	235.3 (92.7)	24.6 (6.0)

Table 2: Proportion of runs stuck in a local optimum (B), and average runtime ($\overline{T}$) and solution size ($\overline{S}$) of successful runs of the RLS-GP using leaf-only substitution and deletion with the complete truth table to evaluate the solution quality for varying n and ℓ. Standard deviations appear in parentheses.
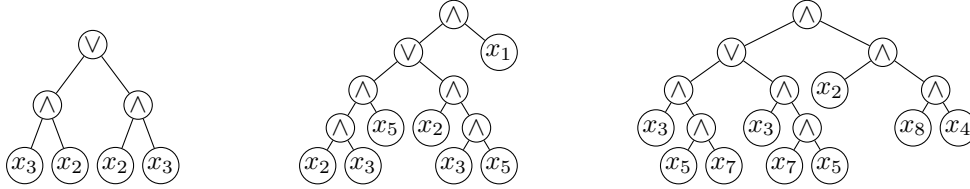

Figure 1: Examples of locally optimal trees, which cannot be improved by substitution or have any single leaf deleted without affecting fitness, constructed by RLS-GP using leaf-only substitution and deletion operations.

Research Question 1: How likely is it that RLS-GP using leaf-only deletion and the complete truth table gets stuck in a local optimum and how does this probability depend on the tree size limit?

Table 2 summarises the experimental behaviour of RLS-GP. The experiments confirm that when using small tree size limits, RLS-GP does indeed get stuck on local optima even on small problem sizes, thus motivating the use of sub-tree deletion. Examples of the locally optimal trees constructed during the runs are depicted in Figure 1. However, the probability of getting stuck decreases as ℓ, the limit on the size of the tree, increases. Concerning the solution quality, with small tree size limits, the number of redundant variables in the final program decreases at the expense of larger runtimes. For $\ell = n$, *'exact'* solutions are returned when the algorithm does not get stuck. On the other hand, larger tree size limits (including no limit) lead to smaller expected runtimes at the expense of redundant variables in the produced programs.

We now turn our attention to the HVL-Prime operator modified to allow subtree deletion, as considered by Theorem 4. The theorem states that the algorithm will find the optimal solution using the complete truth table if an appropriate tree size limit is in place. However, it is unclear from the theory whether the tree size limit is necessary.

Research Question 2: How does the tree size limit affect the runtime and solution quality of RLS-GP using subtree deletion and the complete truth table for evolving the conjunction?

As predicted by the theory, RLS-GP never gets stuck in our experiments when using the complete truth table and a tree size limit. Table 3 shows the average number of iterations required to find the global optimum for various problem sizes and varying tree size limits. Once again the experiments show that smaller tree size limits lead to lower numbers of redundant variables at the expense of a higher runtime. Larger limits, including no limit at all, lead to faster runtimes at the expense of admitting more redundant variables. Noting that in practical applications a tree size limit is often necessary, we leave the proof that the algorithm evolves an exact conjunction without any limits on the tree size for future work.

We now add the negated literals to the terminal set to turn our attention to the performance of RLS-GP when equipped with a complete set of functions and terminals that allow to express any Boolean function. Since it has been proven that negations do not allow to efficiently evolve conjunctions when only the AND function is available to RLS-GP using the complete truth table (Lissovoi and Oliveto, 2019), we have conjectured that the same applies when also the OR function is available to the algorithm.

Research Question 3: Can RLS-GP using the fully expressive set of functions and literals evolve the con-

| n | $\ell = n$ | | $\ell = n+1$ | | $\ell = 2n$ | | $\ell = \infty$ | |
	$\overline{T}$	$\overline{S}$	$\overline{T}$	$\overline{S}$	$\overline{T}$	$\overline{S}$	$\overline{T}$	$\overline{S}$
4	51.2	4.0	42.5	4.4	38.8	5.1	39.1	5.3
	(31.1)	(0.0)	(23.5)	(0.5)	(20.8)	(1.2)	(22.3)	(1.8)
8	147.5	8.0	129.9	8.7	93.5	11.3	92.3	11.6
	(83.3)	(0.0)	(69.1)	(0.5)	(39.1)	(2.4)	(38.1)	(3.0)
12	325.9	12.0	233.4	12.8	153.6	17.7	151.2	18.3
	(184.4)	(0.0)	(123.9)	(0.4)	(56.6)	(3.1)	(50.3)	(3.8)
16	544.6	16.0	377.0	16.9	228.3	24.5	221.0	25.2
	(333.8)	(0.0)	(176.0)	(0.4)	(74.6)	(3.7)	(72.0)	(4.9)

Table 3: Average runtime ($\overline{T}$) and solution size ($\overline{S}$) of RLS-GP using the subtree deletion sub-operation, and the complete truth table to evaluate the solution fitness, for varying n and ℓ. Standard deviations appear in parentheses.

| n | | $\ell = n$ | | | $\ell = n+1$ | |
	B	$\overline{T}$	$\overline{S}$	B	$\overline{T}$	$\overline{S}$
4	0.000	655.3 (638.0)	4.0 (0.0)	0.000	513.7 (480.8)	4.5 (0.5)
8	0.998	1750.0 (-)	8.0 (-)	0.994	5535.3 (3066.0)	8.3 (0.6)
12	1.000	-	-	1.000	-	-
16	1.000	-	-	1.000	-	-

| n | | $\ell = 2n$ | | | $\ell = \infty$ | |
	B	$\overline{T}$	$\overline{S}$	B	$\overline{T}$	$\overline{S}$
4	0.000	431.0 (392.7)	5.7 (1.3)	0.000	422.7 (425.7)	6.4 (2.5)
8	0.990	6443.8 (3789.4)	11.0 (1.6)	0.982	4478.7 (2371.6)	12.4 (3.3)
12	1.000	-	-	1.000	-	-
16	1.000	-	-	1.000	-	-

Table 4: Proportion of runs failing to find the global optimum within 10^4 iterations (B), and average runtime ($\overline{T}$) and solution size ($\overline{S}$) of successful runs of the RLS-GP using the subtree deletion sub-operation, negated literals, and the complete truth table to evaluate the solution fitness, for varying n and ℓ. Standard deviations, where available, appear in parentheses.

junction using the complete truth table?

Table 4 shows the effect of allowing negated literals: as the problem size increases, RLS-GP is overwhelmingly likely to construct a negation and not find the global optimum within the allotted runtime of 10,000 iterations as we have conjectured.

Finally, we examine the behaviour of RLS-GP when using an incomplete training set which allows us to evolve bigger conjunctions i.e., larger problem sizes. The results from Theorems 6 and 7 rely on the algorithm stopping once a logarithmic sampled error is achieved using $F = \{AND, OR\}$ respectively without and with negated literals. However, from the theory it is unclear whether this error threshold is necessary or not.

Research Question 4: Is the logarithmic error threshold required by RLS-GP using a training set to evolve the conjunction and how does removing the threshold affect the algorithm's performance using only positive literals?

We run experiments comparing the performance of RLS-GP when stopping at error 0 or stopping earlier for $n = 50$. The average runtimes of the two variants are plotted in Figure 2. The figure confirms our theoretical result that the algorithms generally run in logarithmic time and produce solutions that contain a logarithmic number of leaf nodes with respect to the training set size. Stopping at 0 error leads to better solutions at the expense of higher runtimes.

We now turn our attention to negated literals.

Research Question 5: Is the logarithmic error threshold required by RLS-GP using a training set to evolve the conjunction and how does removing the threshold affect the algorithm's performance using both positive and negated literals?

Figure 3 explores the effect of expanding the literal set with the negated variables. For $n = 50$, negations cause a contradiction to be constructed when a solution contains approximately 8 leaf nodes on average, allowing the algorithm to terminate within an average of 50 iterations even for larger training sets. Figures 4 and 5 show the average number of ORs in the final program with and without negations in the literal set respectively. While these are few in number, more disjunctions are present the closer the threshold on acceptable sampled error is to

0.

6   Conclusion

We analysed the behaviour of a variant of the RLS-GP algorithm and proved expected runtime bounds when using the complete truth table to evaluate the solution quality, as well as when using a polynomial sample of possible inputs chosen uniformly at random. Equipped with a tree size limit and a mutation operator capable of deleting entire subtrees, RLS-GP is able to efficiently evolve a Boolean function – AND_n, the conjunction of n variables (or OR_n a disjunction of n variables) – when given access to both the binary conjunction and disjunction operators.

When using the complete truth table to evaluate the quality of solutions, we show that in expectation, an optimal solution is found within $O(\ell n \log^2 n)$ iterations if the terminal set contains exactly the distinct literals contained in the target function. Experimentally, we see that the GP system is able to find solutions quicker as ℓ, the limit on the tree size, increases, suggesting that the theoretical bound is overly pessimistic in its modelling of the process. Conversely, solutions with larger tree size limits tend to contain more redundant variables, suggesting a trade-off between optimisation time and solution complexity. However, if unnecessary negated literals are also present in the terminal set, then we show experimentally that the algorithm gets stuck on a local optimum (i.e., a contradiction) with overwhelming probability. We leave a formal proof for future work.

When sampling a polynomial number of inputs to evaluate the program quality, the evolved solutions are not exactly equivalent to the target function, but generalise well: any polynomially small generalisation error can be achieved by sampling a polynomial number of inputs uniformly at random in each iteration. This result holds even if the GP system is completely expressive i.e., it can represent any Boolean function of at most n distinct variables. Our theoretical results predict that RLS-GP is usually able to avoid inserting ORs in this setting, which is reflected in our experimental results. However, when the unnecessary negated literals are present, then a contradiction is inserted in the tree with high probability leading to solutions of smaller average sizes i.e., the OFF function, that always returns false, may actually be evolved. Since it returns the correct output on all inputs but one, it generalises well nevertheless.

While these results represent a considerable step forward for the theoretical analysis of GP behaviour, much work remains to be done. In addition to addressing the open problem of removing the limit on the tree size, the analysis should be extended to cover the evolution of Boolean conjunctions and disjunctions of arbitrary size i.e., not necessarily using all of the n available variables. Furthemore, more complex target functions should be considered where the use of populations and crossover may be essential.

References

Bartoli, A., Cumar, S., Lorenzo, A. D., and Medvet, E. (2014). Compressing regular expression sets for deep packet inspection. In *Proceedings of Parallel Problem Solving from Nature (PPSN XIII)*, pages 394–403. Springer.

Corus, D., Lissovoi, A., Oliveto, P. S., and Witt, C. (2021). On steady-state evolutionary algorithms and selective pressure: Why inverse rank-based allocation of reproductive trials is best. *ACM Transactions on Evolutionary Learning and Optimization*, 1:1–38.

Corus, D. and Oliveto, P. S. (2020). On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms. *Algorithmica*, 82:3676–3706.

Dang, D., Eremeev, A. V., and Lehre, P. K. (2021). Escaping local optima with non-elitist evolutionary algorithms. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pages 12275–12283. AAAI Press.

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

Doerr, B. (2019). Analyzing randomized search heuristics via stochastic domination. *Theoretical Computer Science*, 773:115–137.
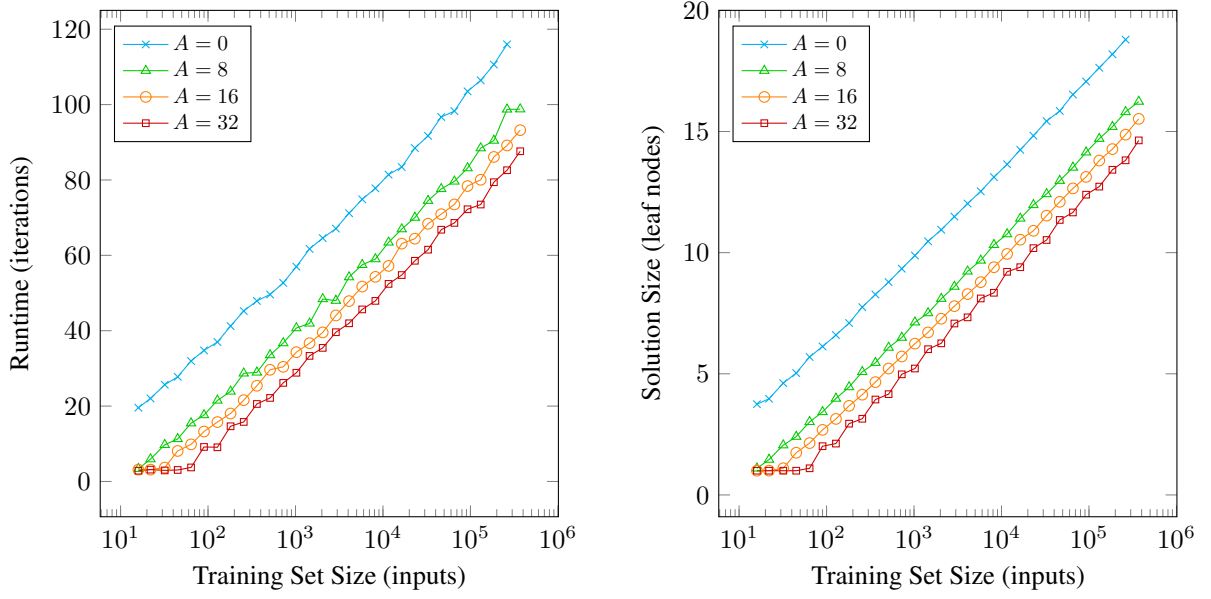
Figure 2: Average runtime and tree size produced by RLS-GP with subtree deletion, using an incomplete training set, stopping once sampled error is at most A, $n = 50, \ell = \infty$, averaged over 500 independent runs.
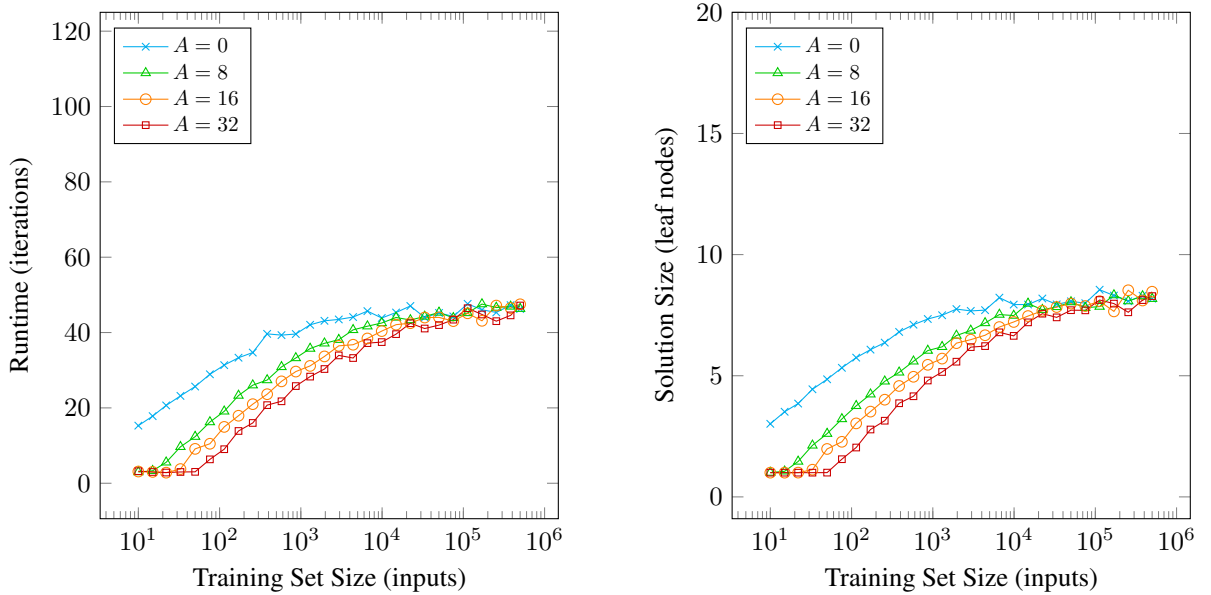

Figure 3: Average runtime and tree size produced by RLS-GP with subtree deletion and negated literals, using an incomplete training set, stopping once sampled error is at most A, $n = 50, \ell = \infty$, averaged over 500 independent runs.
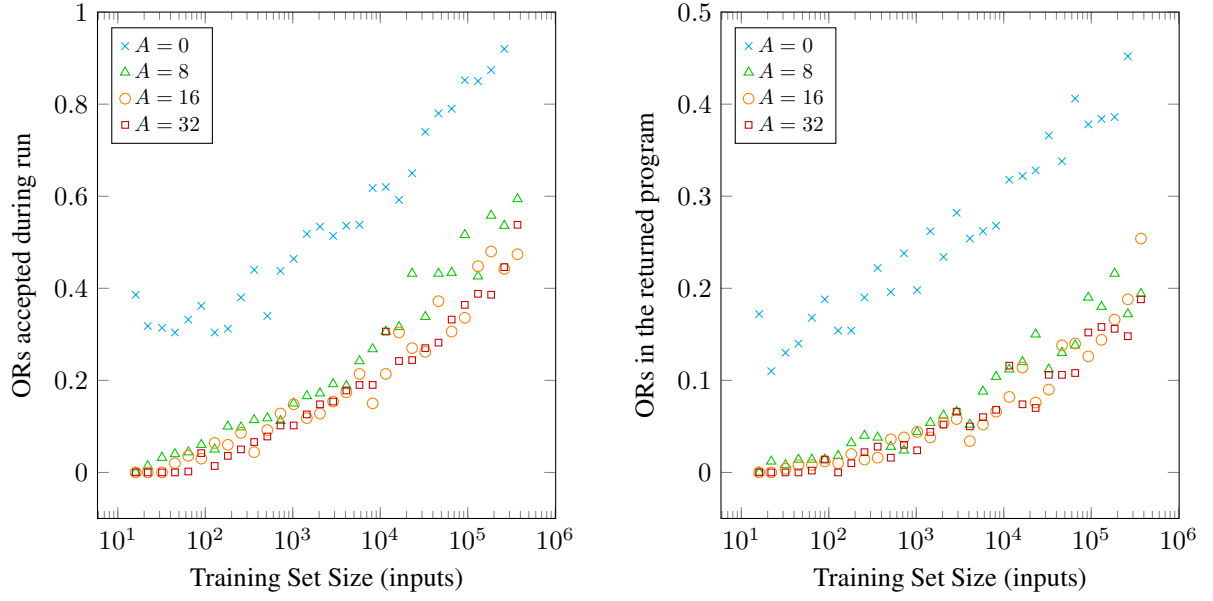
Figure 4: Number of OR nodes inserted and surviving to the solution returned by RLS-GP with subtree deletion, using an incomplete training set, stopping once sampled error is at most A, $n = 50, \ell = \infty$, averaged over 500 independent runs.
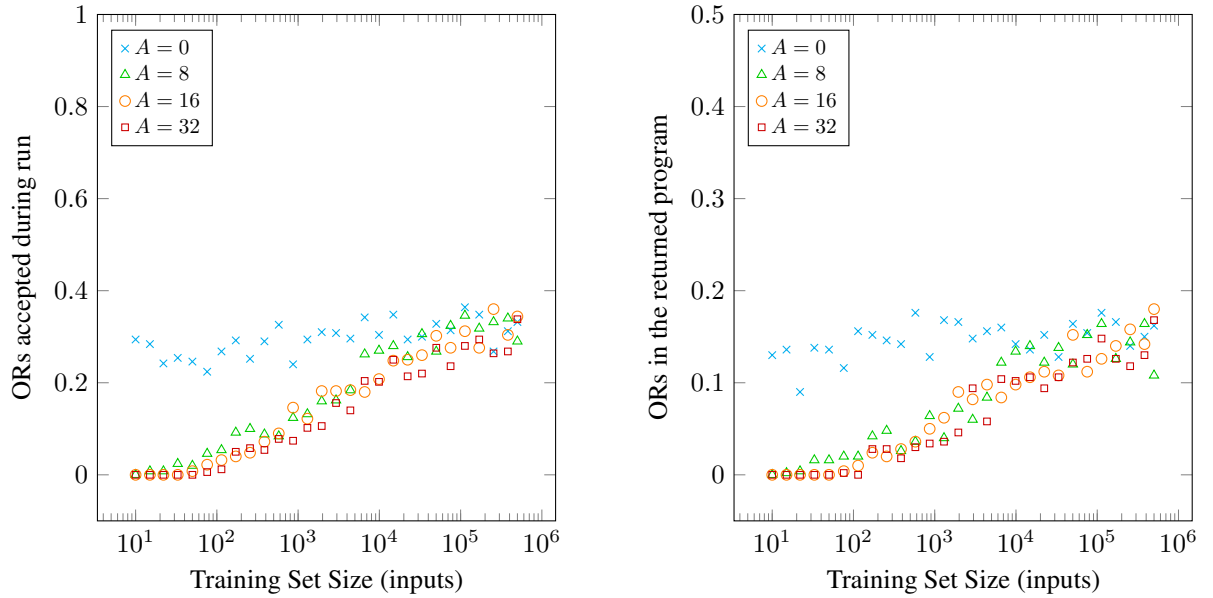

Figure 5: Number of OR nodes inserted and surviving to the solution returned by RLS-GP with subtree deletion and negated literals, using an incomplete training set, stopping once sampled error is at most A, $n = 50, \ell = \infty$, averaged over 500 independent runs.

Doerr, B. (2020). Probabilistic tools for the analysis of randomized optimization heuristics. In Doerr, B. and Neumann, F., editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 1–87. Springer.

Doerr, B. and Goldberg, L. A. (2013). Adaptive drift analysis. *Algorithmica*, 65:224–250.

Doerr, B., Happ, E., and Klein, C. (2012a). Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33.

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

Doerr, B., Kötzing, T., Lagodzinski, J. G., and Lengler, J. (2020). The impact of lexicographic parsimony pressure for order/majority on the run time. *Theoretical Computer Science*, 816:144–168.

Doerr, B., Lissovoi, A., and 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)*, pages 1003–1011.

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

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

Gathercole, C. and Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Proceedings of Parallel Problem Solving from Nature (PPSN III)*, pages 312–321. Springer.

Huang, Z., Zhou, Y., Chen, Z., and He, X. (2019). Running time analysis of MOEA/D with crossover on discrete optimization problem. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 2296–2303. AAAI Press.

Jansen, T., Jong, K. A. D., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440.

Johannsen, D. (2010). *Random combinatorial structures and randomized search heuristics*. PhD thesis, Universität des Saarlandes, Saarbrücken.

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

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

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

Lehre, P. K. and Oliveto, P. S. (2018). Theoretical analysis of stochastic search algorithms. In Martí, R., Pardalos, P. M., and Resende, M. G. C., editors, *Handbook of Heuristics*, pages 849–884. Springer.

Lengler, J. (2020). Drift analysis. In Doerr, B. and Neumann, F., editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 89–131. Springer International Publishing, Cham.

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*, 66:655–689.

Lissovoi, A. and Oliveto, P. S. (2020). Computational complexity analysis of genetic programming. In Doerr, B. and Neumann, F., editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 475–518. Springer International Publishing, Cham.

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

Lynch, D., Fenton, M., Fagan, D., Kucera, S., Claussen, H., and O'Neill, M. (2019). Automated self-optimization in heterogeneous wireless communications networks. *IEEE/ACM Transactions on Networking*, 27(1):419–432.

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

Miranda, I. M., Aranha, C., and Ladeira, M. (2019). Classification of EEG signals using genetic programming for feature construction. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2019)*, pages 1275–1283.

Mitavskiy, B., Rowe, J. E., and Cannings, C. (2009). Theoretical analysis of local search strategies to optimize network communication subject to preserving the total number of links. *International Journal of Intelligent Computing and Cybernetics*, 2(2):243–284.

Moore, J. H., Olson, R. S., Chen, Y., and Sipper, M. (2018). Automated discovery of test statistics using genetic programming. *Genetic Programming and Evolvable Machines*, 20(1):127–137.

Neumann, F. and Wegener, I. (2007). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40.

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

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu Press.

Sutton, A. M. (2021). Fixed-parameter tractability of crossover: Steady-state GAs on the closest string problem. *Algorithmica*, 83:1138–1163.

Vu, T. M., Probst, C., Epstein, J. M., Brennan, A., Strong, M., and Purshouse, R. C. (2019). Toward inverse generative social science using multi-objective genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2019)*, pages 1356–1363.

Witt, C. (2006). Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86.

Zheng, W., Liu, Y., and Doerr, B. (2022). A first mathematical runtime analysis of the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). In *Proceeding of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, pages 10408–10416. AAAI Press.