Runtime Analysis of Cartesian Genetic Programming on MAX: A Proven Exponential Speedup

Duc-Cuong Dang
Chair of Algorithms for Intelligent Systems,
University of Passau, Passau, Germany
duccuong.dang@uni-passau.de

Roman Kalkreuth
Chair for AI Methodology (AIM),
Faculty of Computer Science,
RWTH Aachen University, Aachen, Germany
kalkreuth@aim.rwth-aachen.de

Andre Opris
Chair of Algorithms for Intelligent Systems,
University of Passau, Passau, Germany
andre.opris@uni-passau.de

Abstract

Genetic Programming (GP) is a search paradigm inspired from natural evolution for the automated discovery of expressions, functions, and computer programs. Cartesian GP (CGP) is a flavor of GP that uses a graph-based model to encode candidate programs, as contrasted to the conventional Tree-based GP (TGP). Since its inception two decades ago, CGP has predominantly been analyzed empirically, thus little is known regarding its theoretical performance guarantees.

This paper analyzes CGP for the MAX problem, which has been only rigorously studied for TGP with a proven expected runtime superlinear in the optimal program length n. We prove that the expected time for the (1+1) CGP employing common mutation operators to evolve an equivalent optimal of the same output is polylogarithmic in n, specifically at most $\mathcal{O}(\log^9 n)$. This is remarkable, as it shows an exponential speedup by switching to CGP. Our results shed light on the benefit and the compactness of the graph-based representation for programs, and on how CGP can navigate its search and fitness spaces efficiently. This is a first proven performance guarantee for CGP on an established benchmark problem. Experiments complement our theoretical findings.

1 Introduction

Genetic Programming (GP) is a search paradigm and a subfield of Evolutionary Computation (EC) and Artificial Intelligence (AI) that evolves computer programs. The goal of applying GP is to automate the process of discovering computer programs based solely on given specifications, such as constraints and input-output examples. Early work by Cramer [7] and Dickmanns et al. [12] applied Genetic Algorithms (GA) to program generation. Koza [42] later introduced tree-based GP (TGP) to evolve LISP S-expressions, and with follow-up work [43] established the research field.

In tree-based GP (TGP), or "Koza-style GP", programs are represented as trees: leaves are input variables or constants from a terminal set T, internal nodes are compute functions from a set F, and the root produces the output. Since then, GP has been extended to include different flavors, depending on how programs are encoded, i. e. having different search spaces, for example linear-sequence [63, 66], graph-based [57, 68], rule-based [75] models, or on how they are evolved, i. e. using different search operators, for example semantically driven designs [59]. Recently, AI-related techniques have been used to enhance GP. Components from Neural Architecture Search (NAS),

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

such as transformers [1] and autoencoders [83], have been integrated into GP, while Large Language Models (LLMs) have been used as search operators for evolving program code [32, 46, 61, 73].

Despite these recent high-profile advances, GP remains largely experimental, with most rigorous theoretical results confined to TGP [51]. In particular, the tradeoffs associated with non tree-based search spaces are not well understood. In a survey of GP research, O'Neill et al. [62] also identified understanding GP representations for programs, and subsequently finding the appropriate ones for problems, as a highly challenging open issue. Motivated by these gaps, we study Cartesian Genetic Programming (CGP), a popular and non tree-based flavor of GP introduced by Miller [57]. CGP uses a graph-based representation with a fixed length vector encoding of programs (see Section 3). Although introduced two decades ago and successfully applied in many areas such as NAS [78], logic synthesis [27, 76], policy learning [82], cryptography [67], and image processing with human competitive results [6], very little is known regarding its theoretical guarantees.

Our research questions are: what advantages does the graph-based representation of CGP provide, and how efficiently can CGP explore its search space for a given fitness function? To answer these, we apply runtime analysis [2, 13, 35]. Here, GP is treated as a randomized algorithm, and performance is measured by the expected number of fitness evaluations to evolve a desired program. Efficiency corresponds to polynomial growth, whereas exponential growth indicates inefficiency.

Our contribution: We present a first runtime analysis of CGP on the MAX problem, specifically MAX-depth-D-$\{\times, +\}$-$\{t\}$, a standard TGP benchmark. The goal is to evolve a program that maximizes the output for bounded depth D using only addition and multiplication, with a single constant $t \geq 1$. We show that the optimal TGP program of size $n = 2^{D+1} - 1$ can be encoded in CGP with a vector of length linear in $D = \Theta(\log n)$. We then prove that the $(1+1)$ CGP with common mutation operators, including single active gene mutation (SAM) [30], finds its optimal program in expected time[1] $\mathcal{O}(\log^9 n)$ for $t \in [1, 2)$, improving to $\mathcal{O}(\log^8 n)$ for $t \geq 2$. Since the $(1 + 1)$ TGP requires superlinear time in n for the same task [40], this implies an exponential speedup. This answers our first research question on the benefit of the graph-based representation of programs that CGP brings.

Our analysis follows the progress in fitness of the evolving CGP program. Particularly, we identify which location and elements in the vector-encoded CGP program that can be changed to make a significant improvement to fitness and with a sufficient probability. This leads to the polynomial time bounds and answers our second research question, that is, CGP can navigate its search space efficiently by steadily increasing the fitness of the current program. Theoretical research of EC algorithms is predominantly done on the search space of bit strings, including recent cutting edge results, e. g. [47, 49, 65]. Our work is the first to analyze integer-valued vectors that encode acyclic graphs representing computer programs. In particular, the components of the vectors are bounded, they can be active or inactive, and model a mix of ordinal variables (encoding nodes in a chain) as well as categorical ones (encoding indices of a lookup table, i. e. see Section 3). Finally, our experiments conducted on MAX for both TGP and CGP complement the theoretical findings.

2 Related work

Runtime analysis has largely contributed to our understanding of the performance of EC algorithms, e. g. see [2, 13] for surveys and [4, 48, 49, 64, 65] for some recent progresses. It has also led to the development of new algorithms and operators that not only have proven runtime guarantees for artificial benchmarks, but also perform competitively on practical problems [9, 10, 18, 23, 31, 47].

Early theoretical work on TGP focused on schema theory [69, 70, 71]. The first runtime analysis appeared in [60] and followed by [20, 41, 81], analyzing the MAJORITY and ORDER problems to evolve structured trees (non-executable programs). The MAX problem was introduced in [29] and studied in [44, 45] for crossover-only GP algorithms. Runtime results of MAX appeared in Kötzing et al. [39, 40], indeed the first to prove performance guarantees for evolving executable programs. They showed that the expected runtime of simple (1+1) TGP systems using mutation operators based on the so-called HVL-prime operator (see Section 4 or [60]) is $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$, and proved some benefit for biased mutations. Mambrini and Oliveto [56] were the first to rigorously analyze GP for evolving Boolean functions. They showed that the same TGP systems are

[1] Here time bounds are expressed in n for literature comparison, later in our results they will be written in D.

efficient for AND_n but not for XOR_n, reflecting the fact that XOR_n is PAC-learnable [79] but not evolvable [80]. These results motivated further work under more realistic settings [22, 50].

Mathematical analysis of GP on the other search spaces of programs is scarce. The only studies that we know of are Kalkreuth and Droschinsky [36, 37], which proposed the first rigorous analysis for the $(1+1)$-CGP with SAM mutation. They showed an expected runtime bound $\Theta(n \log n)$ for the SUM problem, and bounds $\Omega(n^2)$ and $\mathcal{O}(n^2 \log n)$ for the COUNTINGOPERATORS problem. However, only one computing node was defined in those problems; thus it is unclear how the results can be generalized for a large number of nodes. CGP programs are encoded as a vector of integers of diverse ranges. The rigorous analysis of Evolutionary Algorithms (EAs) on integer-valued problems was initiated by Doerr et al. [15], and was further explored in [14] on the benefit of high mutation rates, in [19, 21] for self-adaptation, and in [72] for the multiset selection problem. Analysis of unbounded integer-valued problems was also considered, e. g. [24, 74]. It remains open how the insights gained from those studies can be useful for CGP where the vector encodes a graph.

Theoretical work on EC algorithms for problems on graphs can be found in [3, 5, 11, 16, 17, 77]. The original form of CGP was inspired by FPGA circuits [57] and proposed to use a 2D grid topology for the graphs with a level-back parameter, thus sharing some similarities with Neural Networks (NNs). However, in NNs the graph topology and functionality on nodes are fixed, then back propagation and numerical optimization are used to compute the edge weights [8, 34]. In contrast, CGP optimizes both the topology and functionality on nodes in parallel using evolutionary search. Recent works on NNs and graph learning can be found in [28, 33, 52, 84]. There has been a recent interest in runtime analysis of Evolutionary NAS (ENAS), i. e. see Fischer et al. [25, 26] and Lv et al. [53, 54, 55].

3 Cartesian Genetic Programming

CGP is a popular representative of graph-based GP. It is defined for N_{in} inputs, N_{out} outputs, and N_{func} (internal) function nodes and uses a lookup table of K functions of arity most k. A (valid) program for such a CGP is represented by a directed acyclic graph $G = (V, E)$, called the *phenotype*, that tells how the outputs are computed from the inputs by composing lookup functions (see Figure 1). Inputs and the lookup table correspond to the terminal set T and function set F in TGP.

We follow the modern approach for CGP in which function nodes are arranged on one line. Nodes are indexed by $V = [N-1]_0$, where $N := N_{\text{in}} + N_{\text{func}} + N_{\text{out}}$, ordered as inputs, function nodes, then outputs. Here $[n]$ for $n \in N$ is the notation for $\{1, \dots, n\}$ and $[n] := [n] \cup \{0\}$. Input nodes have no incoming edges, while each function node has k incoming edges from nodes of smaller index, encoding ordered arguments (some edges are ignored if the arity of the lookup function is smaller than k). Each output node has one incoming edge from a non output node.

The graph G is encoded as a vector of $N_{\text{func}}(k+1) + N_{\text{out}}$ integers, called the *genotype*. The vector is divided into blocks indexed from N_{in} to $N-1$, matching the indices of the nodes in phenotype that they encode. A block with index $i < N_{\text{in}} + N_{\text{func}}$ (a function block) contains $k+1$ integers, the first, called a *function gene*, is an integer in $[K-1]_0$ (an index in the lookup table) while the rest, called *connection genes*, are integers in $[i-1]_0$. A block of larger index i (an output block) only contains an integer in $[N - N_{\text{out}} - 1]_0$, called an *output connection gene*. The set $\mathcal{X}$ of all genotypes (all valid CGP programs) is a space of integer-valued vectors.

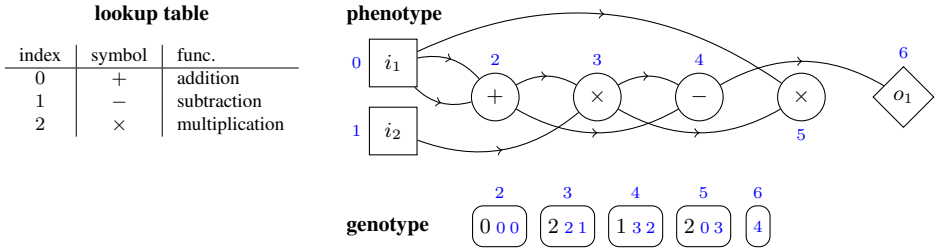


Figure 1: Example of a CGP program that computes the expression $o_1 = (i_1 + i_1) \times i_2 - (i_1 + i_1)$. Node 5 is a so-called inactive node because no paths connect it to the output.

We consider the simplest CGP algorithm given in Algorithm 1, the so-called (1+1) CGP. Starting from a first program initialized uniformly at random, in each iteration a new y offspring program is generated by mutating the current one. The offspring will replace the parent if it has better or equally good fitness. Here $f(x)$ denotes the fitness of a program x, which is problem-dependent and assumed to be given. By convention in runtime analysis, we let the algorithm run infinitely then estimate the expected time to reach some target state. Practical CGP algorithms only use a slightly more general framework, where λ (typically 4) offspring programs like y are generated per iteration.

Algorithm 1: The (1+1)-CGP on the space $\mathcal{X}$ of programs.

$x \sim \mathrm{Unif}(\mathcal{X})$
for iter $:= 1 \to \infty$ **do**
  $y := \mathrm{mutate}(x)$
  if $f(y) \geq f(x)$ **then**
    $x := y$

Algorithm 2: NEWVALUE(x, i).

$b := \lfloor i/(k+1) \rfloor$ // block index
$j := i \mod (k+1)$ // index inside a block
if $i < N_{\mathrm{func}}(k+1)$ **then**
  if $j = 0$ **then** $S := [K-1]_0$ // valid function genes
  else $S := [N_{\mathrm{in}} + b - 1]_0$ // valid input connection genes
else
  $S := [N_{\mathrm{in}} + N_{\mathrm{func}} - 1]_0$ // valid output connection genes
if $|S \setminus \{x[i]\}| \geq 1$ **then**
  $S := S \setminus \{x[i]\}$ // avoid the old value if possible
return $v \sim \mathrm{Unif}(S)$

CGP mutations: Mutation operators can choose to change any gene (component) of the genotype (vector) to a new value uniformly at random while respecting the constraint on their range. Such a change is done by the procedure NEWVALUE which is shown in Algorithm 2.

Two mutation operators are common in CGP. The first, probabilistic mutation (Algorithm 3), mutates each gene independently with probability p, generalizing standard bit mutation on bit strings to the CGP search space $\mathcal{X}$. The second, single active gene mutation (SAM) [30] (Algorithm 4), repeatedly mutates genes chosen uniformly at random until one associated with a so-called *active* node is modified. A node is active if it lies on a path to an output node in the phenotype. All output nodes are therefore active by this definition. Active nodes can be identified during evaluation, thus we assume having access to an ISACTIVE function.

Algorithm 3: MUTATIONPROB(x, p).

Copy x to y
for $i := 0 \to N_{\mathrm{func}}(k+1) + N_{\mathrm{out}} - 1$ **do**
  $r \sim \mathrm{Unif}((0,1))$
  if $r < p$ **then** $y[i] :=$ NEWVALUE(y, i)
return y

Algorithm 4: MUTATIONSAM(x).

Copy x to y
active := false
while ¬active **do**
  $i \sim \mathrm{Unif}([N_{\mathrm{func}}(k+1) + N_{\mathrm{out}} - 1]_0)$
  $y[i] :=$ NEWVALUE(y, i)
  active := ISACTIVE$(x, i) \wedge (y[i] \neq x[i])$
return y

A transfer lemma: We will focus on SAM mutation as it is parameter-free and is used more often in practice. However, upper bound results are asymptotically transferable to the other operator as follows. Formally, a mutation operator mut is described by a transition matrix pmut where each entry is $\mathrm{pmut}(x, y) := \Pr(\mathrm{mut}(x) = y)$ for a pair $x, y \in \mathcal{X}$ of programs. Thus we have the matrices $\mathrm{pmut}_{\mathrm{SAM}}$ and $\mathrm{pmut}_{\mathrm{PROB}}$ respectively. The probabilistic mutation is a so-called *global mutation* since $\mathrm{pmut}_{\mathrm{PROB}}(x, y) > 0$ for all $x, y \in \mathcal{X}$. In contrast for SAM mutation, $\mathrm{pmut}_{\mathrm{SAM}}(x, y) > 0$ requires that y modifies exactly one active node of x and possibly some inactive ones.

For x, y satisfying $\mathrm{pmut}_{\mathrm{SAM}}(x, y) > 0$, it is often non-trivial to compute $\mathrm{pmut}_{\mathrm{SAM}}(x, y)$. But we can estimate a lower bound $\mathrm{pmut}'_{\mathrm{SAM}}(x, y)$ for it, which is defined as follows. Let $L := N_{\mathrm{func}}(k+1) + N_{\mathrm{out}}$ be the genome length, M be the set of genes where x and y differ, and for each $s \in M$, let $q(s)$ be the probability that NEWVALUE(x, s) returns $y[s]$. We then fix a specific order of the genes of M for the while-loop in SAM to modify in x so that the gene associated with the active node is modified last. This order corresponds to one specific way to reach y from x, and since in each iteration of the loop, a gene is selected uniformly at random (probability $1/L$) to be modified, thus $\mathrm{pmut}_{\mathrm{SAM}}(x, y) \geq \prod_{s \in M}(1/L)q(s) =: \mathrm{pmut}'_{\mathrm{SAM}}(x, y)$. For x, y where $\mathrm{pmut}_{\mathrm{SAM}}(x, y) = 0$, we define $\mathrm{pmut}'_{\mathrm{SAM}}(x, y) := 0$. The following lemma, allowing the transfer of results, shows that the (1+1) CGP using the probabilistic mutation can follow the same path as the algorithm using SAM

mutation under the same initialization. The probabilities of these two walks are asymptotically equal if that of the former event was estimated using $\mathrm{pmut}'_{\mathrm{SAM}}$ and the path length is constant.

Lemma 1. *Let $L := N_{\mathrm{func}}(k+1)+N_{\mathrm{out}}$. Consider a sequence of $k+1$ CGP programs $x_0,\dots,x_k$ such that for all $i \in [k]$, $f(x_{i-1}) \le f(x_i)$ and $\mathrm{pmut}'_{\mathrm{SAM}}(x_{i-1},x_i) \ge p_i > 0$. Then after k iterations the (1+1) CGP using SAM mutation and initialized with program x_0 has probability of at least $\tilde{p} := \prod_{i=1}^{k} p_i$ to change its current program to x_k. Moreover, the (1+1) CGP using the probabilistic mutation with parameter $p = \frac{1}{L}$ under the same condition can reach program x_k with probability at least $\tilde{p}\cdot e^{-k}$, which is $\Omega(\tilde{p})$ if $k = O(1)$. Here the asymptotic notation is w. r. t. L.*

Compactness of CGP programs: A prerequisite for the efficiency of CGP is that its programs should not be too large. Unlike in TGP, where a value computed at a node can only be fed to one parent, in CGP, the value can be fed to any subsequent nodes. Therefore, a CGP program can inherently leverage reusable subprograms. This fact is well-known in the GP community [58], and we introduce the following lemma to formalize the idea.

Lemma 2. *For every TGP program x that uses functions from a set F for its internal nodes $I(x)$ and elements of a terminal set T for its leaves, there exists an equivalent CGP program z that produces the same output as x and uses T inputs, F in the lookup table, and at most $|I(x)|$ function nodes. Moreover, if there exist $y_1, y_2 \in I(x)$ such that the subtrees rooted at these nodes are disjoint and identical (reusable subprograms), then z only requires at most $|I(x)| - |I(y_1)|$ function nodes.*

4 The Max Problem

The MAX problem was introduced in [29], empirically studied in [44, 45], and analyzed theoretically in [39, 40]. In MAX-depth-D-F-T, the goal is to construct a program of maximum output value using a function set F of binary functions, a terminal set T of constants, and a maximum tree depth of D. A valid TGP program has at most $2^D - 1$ internal nodes and 2^D leaves, hence at most $n := 2^{D+1} - 1$ nodes in total. The analysis of MAX in [39, 40] studied mutation-only TGPs based on the HVL-prime operator [60]. Each application of HVL-prime on a TGP program chooses uniformly at random one of three operations to execute: (i) substitute a function node, (ii) insert a function at a leaf, or (iii) delete a subtree. In the (1+1) TGP-single variant, HVL-prime is applied once per mutation; in (1+1) TGP-multi, it is applied k times, where $k \sim \mathrm{Poisson}(1)$. The (1+1) search strategy is Algorithm 1 but on the search space $\mathcal{X}$ of TGP programs, and all trees with depth exceeding D have fitness $-\infty$. The authors of [40] derived the following bounds on the optimization time, and a matching lower bound of $\Omega(n \log n)$ follows via a coupon collector argument.

Theorem 3 (Theorems 5, 6, 7 and 10 in [40])**.** *The expected optimization time of the (1+1) TGP-single algorithm on* MAX*-depth-D-$\{\times,+\}$-$\{t\}$ is $\mathcal{O}(n \log n)$ for all $t > 0$, while that of (1+1) TGP-multi on* MAX*-depth-D-$\{\times,+\}$-$\{1\}$ is $\mathcal{O}(n^2)$.*

All bounds are exponential in D since $n = 2^{D+1} - 1$, highlighting a key drawback of TGP, that its programs can be huge. Every TGP program has an equivalent CGP representation that can often be compressed. For MAX-depth-D-$\{\times,+\}$-$\{t\}$, CGP uses one input ($N_{\mathrm{in}} = 1$ with $t > 0$), function set $\{\times,+\}$ ($K = 2$, $k = 2$), D function nodes ($N_{\mathrm{func}} = D$), and one output ($N_{\mathrm{out}} = 1$). The setting of N_{func} follows by reducing an optimal symmetric TGP program via Lemma 2, e. g. see Figure 2 for $D = 4$. The genome length of CGP for MAX is therefore $L = N_{\mathrm{func}}(k+1) + N_{\mathrm{out}} = 3D + 1$.

We restrict our analysis to $t \ge 1$. In TGP, [40] showed that t does not affect $(1+1)$ TGP-single, while $(1+1)$ TGP-multi assumes $t = 1$. For $t \in [1, 2)$, in the optimal CGP program, the first node is addition and the rest multiplication, with sequential wiring to the output; for $t = 2$ the first node may vary, and for $t > 2$ it must be multiplication, otherwise the structure is unchanged.

Notation in our analysis: We often consider a specific CGP program in our analyses, e. g. the current solution of the (1+1) CGP. When this is clear from the context, we then focus on its function nodes. Function and input nodes are referred to directly by their node indices in the phenotype (e. g. blue numbers in Figure 2). For a function node $i \in [D]$, we use $i.\mathrm{in}_1, i.\mathrm{in}_2$ and $i.f$ to denote the nodes (again by indices) that its first and second input edges are pointed to and its selected function of the lookup table. When we say setting $i.\mathrm{in}_1 = j$ for $j < i$, we mean redirecting the first input of node i to point at node j. The output value of node i is denoted $\mathrm{val}(i)$ and the domain of $\mathrm{val}(i)$ is denoted $\mathcal{V}(i)$. The input t serves as the base domain and value, i. e. $\mathcal{V}(0) = \{t\}, \mathrm{val}(0) = t$.

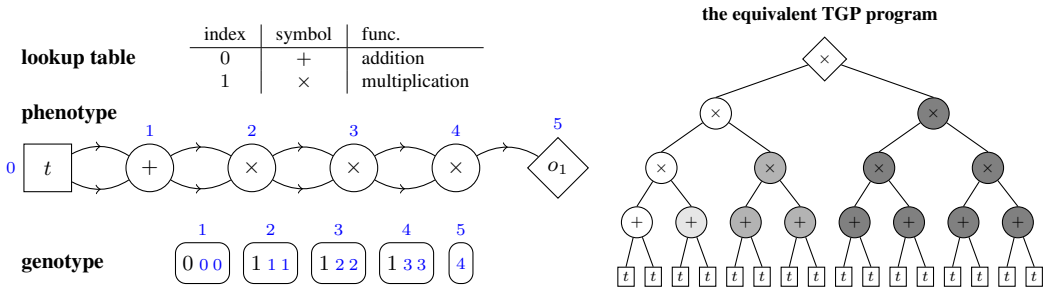


Figure 2: The optimal TGP program (right) of MAX-depth-$\{4\}$-$\{\times,+\}$-$\{t\}$ for $t \in [1,2)$. Gray internal nodes are discarded when converting to an equivalent CGP program (left) by Lemma 2.

Maximum output: The lemma below shows how the set $\mathcal{V}(i)$ grows as i increases for MAX-depth-D-$\{\times,+\}$-$\{t\}$ with $t \geq 1$, and that the maximum output is $\max\{2t,t^2\}^{2^{D-1}}$ as $\max\{2t,t^2\}$ is the maximum of $\mathcal{V}(1)$, which is subsequently amplified by the $D-1$ follow-up multiplications.

Lemma 4. *Consider any CGP program for* MAX*-depth-D-$\{\times,+\}$-$\{t\}$ with $t \geq 1$, $D \in \mathbb{N}$. For any function node $i \in [D]$, the domain $\mathcal{V}(i)$ is defined by the following recurrence relation.*

$$\mathcal{V}(i) = \begin{cases} \{t\} & \text{if } i = 0, \\ \{u+v, u \cdot v \mid u,v \in \bigcup_{k=0}^{i-1} \mathcal{V}(k)\} & \text{if } 1 \leq i \leq D. \end{cases}$$

Consequently, it holds that: (i) $\max\{\mathcal{V}(i-1)\} < \max\{\mathcal{V}(i)\}$ *for all* $i \in [D]$*;* (ii) $\min\{\mathcal{V}(i-1)\} \geq t$ *for all* $i \in [D+1]$*;* (iii) *and the maximum output is* $\max\{\mathcal{V}(D)\} = \max\{2t,t^2\}^{2^{D-1}}$.

Fitness landscape: The fitness landscape of CGP on MAX-depth-D-$\{\times,+\}$-$\{t\}$ is not unimodal, that is, there are programs without a direct improved Hamming neighbor, e. g. see Figure 3. Thus the dynamics of CGP on MAX is non-trivial. Such programs can be used to derived lower bounds for CGP, but as being non-essential to our message, those results are deferred to the appendix.

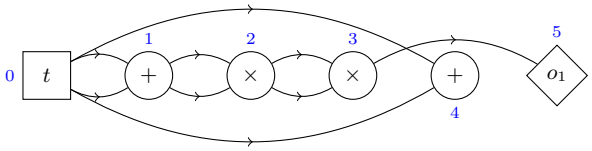


Figure 3: A solution where at least two connection changes are required to increase the output value.

5 General Upper Bound

We now provide the main result of our paper, which is upper bounds for the expected runtime of (1+1) CGP on the MAX problem for any initialization. The challenge in analyzing the dynamics of the current program while optimizing the MAX problem is to come up with a measure of the progress towards an optimal program. It is particularly difficult to come up with a structural measure because the output connection can be be moved around by mutation, i. e. in favor of a fitness improvement elsewhere. Hence, there is always a chance that some nodes of the evolved structure so far become inactive and are destroyed in subsequent mutations.

To overcome this challenge, our strategy for showing an upper bound for the expected runtime of CGP on MAX is to consider fitness improvement. In particular, if the algorithm is executed for a sufficiently long time, there will be a good chance of a significant fitness improvement by a factor of $\min\{a^{1/3}, \max\{2t,t^2\}^{2^{D-1}}/a\}$ if the current fitness $f(x_0)$ is a.

Lemma 5. *Consider the (1+1) CGP using SAM mutation to solve* MAX*-depth-D-$\{\times,+\}$-$\{t\}$ for any $t \geq 1$ and $D \geq 2$. If the current program x_0 has the output value $\max\{2t,t^2\} \leq f(x_0) := a < \max\{2t,t^2\}^{2^{D-1}}$, then there always exists a sequence of programs $x_0,\ldots,x_k$*

with $k = O(1)$*,* $f(x_{i-1}) \leq f(x_i)$ *for all* $i \in [k]$*,* $f(x_k) \geq \min\left\{\max\{2t, t^2\}^{2^{D-1}}, a^{4/3}\right\}$*, and* $\tilde{p} := \prod_{i=1}^{k} \mathrm{pmut}'_{\mathrm{SAM}}(x_{i-1}, x_i)$ *with* $\tilde{p} = \Omega(D^{-8})$ *if* $t \in [1, 2)$ *and* $\tilde{p} = \Omega(D^{-7})$ *if* $t \geq 2$*. Therefore, after at most* $\mathcal{O}(D^8)$ *iterations for* $t \in [1, 2)$ *and* $\mathcal{O}(D^7)$ *for* $t \geq 2$ *in expectation, the output is either already maximized or at least* $a^{4/3}$*.*

We will explore the proof of this key lemma in the next section. Once proven, applying Lemma 5 $\ell := \lceil \frac{D-1}{\log(4/3)} \rceil = \mathcal{O}(D)$ times implies that the (1+1) CGP using SAM mutation, and starting from a program with fitness $a \geq \max\{2t, t^2\}$, will reach a fitness of at least $a^{(4/3)^\ell} \geq \max\{2t, t^2\}^{2^{D-1}}$, i. e. that of an optimal program, in $\mathcal{O}(\ell \cdot D^8) = \mathcal{O}(D^9)$ and in $\mathcal{O}(\ell \cdot D^7) = \mathcal{O}(D^8)$ for $t \in [1, 2)$ and $t \geq 2$ respectively. This is because $(4/3)^\ell \geq (4/3)^{\frac{D-1}{\log(4/3)}} = \left((4/3)^{\log_{4/3} 2}\right)^{D-1} = 2^{D-1}$ and $a \geq 2$. Reaching the initial fitness $\max\{2t, t^2\}$ is easy as it suffices to utilize the first function node, and the expected waiting time is dominated by the mentioned bounds. Those bounds are then transferred to that of the probabilistic mutation by Lemma 1, and constitute our main results.

Theorem 6. *The (*1+1*) CGP using either SAM mutation or the probabilistic one with parameter* $p = 1/(3D + 1)$ *and initialized with any program solves* MAX*-depth-*D*-*$\{\times, +\}$*-*$\{t\}$ *for any* $t \in [1, 2)$ *in at most* $\mathcal{O}(D^9)$ *iterations, and this bound is dropped to* $\mathcal{O}(D^8)$ *for* $t \geq 2$*.*

6 Time to a Significant Improvement

We now present the proof of the key lemma (Lemma 5). It requires a tedious case analysis and is split into several sublemmas corresponding to different settings of the initial solution. To convey the core idea, we will focus on explaining the case separation, and only present the complete proof for one critical setting (Lemma 9) while those for the remaining cases can be found in the appendix. We will state each setting directly and assume the same base conditions as in the key lemma. In particular, parameter a is referred to as the fitness of the initial solution $f(x_0)$ and is assumed $a \geq \max\{t^2, 2t\}$.

The analysis of the critical setting demonstrates how the other proofs can be done. In those, further subcase divisions may be required, and in each subcase, to argue about the probability of a significant improvement, i. e. by a factor of at least $a^{1/3}$, we often consider the worst case that some constant number of calls to SAM mutation is required to achieve some step-by-step goal. For example, a step may require to redirect an input connection of a specific function node to another specific node, i. e. probability at least $1/(3D + 1)$ to select the desired component in the encoded genotype then probability at least $1/(D - 1)$ to change its value correctly as Algorithm 2 excludes the old value when sampling a new one, so probability $\Omega(D^{-2})$ for such a step. This can be improved to $\Omega(D^{-1})$ for function nodes located near the beginning, such as node 1 or 2, since making the correct change then occurs with a constant probability. In all cases, we only reply on bound estimates for $\mathrm{pmut}'_{\mathrm{SAM}}$.

Proving Lemma 5 requires identifying genome positions where large improvements occur. We begin with the easiest case where the last function node D is inactive and can be used to increase the fitness.

Lemma 7. *If the function node* D *is inactive then a single application of SAM mutation has a probability of at least* $\Omega(D^{-7})$ *to increase the fitness by a factor of* a*.*

It is less trivial to identify a location for a significant improvement when node D is active. We found that it is around the following node. Let j^* be the first function node (i. e. the smallest index) such that for all later function nodes $\ell > j^*$ (if they exist) it holds $\ell.\mathrm{in}_1 = \ell.\mathrm{in}_2 = \ell - 1 \wedge \ell.f = \times$. The intuitive reason is that from this node, we can easily track the fitness. More generally, for any function node $\ell > j^*$ we have $\mathrm{val}(\ell) = w^{2^{\ell - j^*}}$, where w is the output value of node j^*.

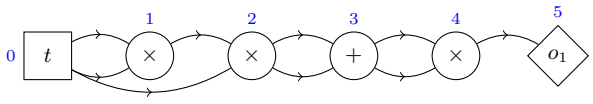


Figure 4: An example solution where $j^* = 3$.

Figure 4 gives an example on how to identify j^*. Note that by definition, if $D.f = +$ then $j^* = D$. Otherwise, when j^* is small enough, SAM can jump directly to the optimal program.

Lemma 8. *If function node D is active and $j^* \leq 3$ then a sequence of 7 applications of SAM mutation has a probability of at least $\Omega(D^{-7})$ to optimize* MAX*-depth-D-$\{\times,+\}$-$\{t\}$.*

For larger j^*, we need to break the cases based on how the node j^* selects its function from the lookup table. Hereafter, we use the following notation for j^*: $n_u := j^*.\mathrm{in}_1$, $n_v := j^*.\mathrm{in}_2$, $u := \mathrm{val}(j^*.\mathrm{in}_1)$, and $v := \mathrm{val}(j^*.\mathrm{in}_2)$, and assume w. l. o. g. that $u \leq v$.

We first consider that an addition is selected, therefore, we have $\mathrm{val}(j^*) = u + v$ and that $a = \mathrm{val}(D) = (u+v)^{2^{D-j^*}} \leq (2v)^{2^{D-j^*}}$. A subcase of this analysis, i. e. u, v are small, leads to a critical separation in the result between $t \in [1,2)$ and $t \geq 2$.

Lemma 9. *If function node D is active, $j^* > 3$, and $j^*.f = +$ then a sequence of 6 applications of SAM mutation has a probability of at least $\Omega(D^{-8})$ to increase the fitness by a factor of $a^{1/3}$ if $t \in [1,2)$ and the probability is improved to $\Omega(D^{-5})$ if $t \geq 2$.*

Proof. If $v \geq 4$, we argue that SAM only needs to perform the following changes (if required) in order: (i) setting $j^*.\mathrm{in}_1 = n_v$ (probability at least $1/((3D+1)(D-1))$); (ii) then changing $j^*.f$ to $\times$ (probability $1/(3D+1)$). These two steps in order cannot decrease the fitness, since the outputs of nodes $1,\ldots,(j^*-1)$ do not change while the outputs of nodes $j^*,\ldots,D$ cannot decrease. This requires at most 2 calls to SAM as node j^* is active, and the success probability is $\Omega(D^{-3})$.

It follows from $v \geq 4$ that $v/2 \geq (2v)^{1/3}$, so a success will lead to an improvement in fitness by a factor of at least

$$\frac{(v^2)^{2^{D-j^*}}}{a} \geq \frac{\left(v^{2^{D-j^*}}\right)^2}{(2v)^{2^{D-j^*}}} = \left(\frac{v}{2}\right)^{2^{D-j^*}} \geq ((2v)^{1/3})^{2^{D-j^*}} \geq a^{1/3}.$$

Otherwise if $v < 4$, and assume $t \in [1,2)$. Then it is sufficient for SAM to use the two first function nodes 1, 2, together with j^* to increase the fitness. The changes (if required) in order are: (i) setting $2.\mathrm{in}_1 = 2.\mathrm{in}_2 = 1$ (probability $1/(3D+1)^2$); (ii) correcting $1.f$ to increase the output of $1.f$ (probability $1/(3D+1)$) (iii) setting $j^*.\mathrm{in}_1 = j^*.\mathrm{in}_2 = 2$ (probability at least $1/((3D+1)(D-1))^2$) then setting $j^*.f$ to $\times$ (probability $1/(3D+1)$). Note that these steps in order do not decrease the fitness, since the outputs of all nodes do not decrease. Hence, those steps are feasible for SAM. At most 6 calls to SAM are needed and the success probability is $\Omega(D^{-8})$.

If successful, we get $\mathrm{val}(1) \geq 2$ thus $\mathrm{val}(2) \geq 4$ regardless of whether $2.f$ is $\times$ or $+$ and so $\mathrm{val}(j^*) \geq 16$. The fitness is then increased by a factor of at least

$$\frac{16^{2^{D-j^*}}}{(2v)^{2^{D-j^*}}} > \frac{16^{2^{D-j^*}}}{8^{2^{D-j^*}}} = \left(8^{2^{D-j^*}}\right)^{1/3} > \left((2v)^{2^{D-j^*}}\right)^{1/3} \geq a^{1/3}.$$

Finally, when $t \geq 2$ we already have $\mathrm{val}(1) \geq 2$ regardless of what $1.f$ is. Thus it is sufficient for SAM to directly use function node 1 to achieve the same fitness because $\mathrm{val}(j^*) \geq 4^2 \geq 16$. This increases the success probability to $\Omega(D^{-5})$. □

We now consider that $j^*.f$ is $\times$, thus $a = (uv)^{2^{D-j^*}} \leq (v^2)^{2^{D-j^*}}$. Here if the preceding function node j^*-1 is inactive, then the latter can be used to increase the fitness by a factor $a^{1/2} > a^{1/3}$.

Lemma 10. *If function node D is active but j^*-1 is not, and $j^* > 3$ and $j^*.f = \times$, then a single application of SAM mutation has a probability of $\Omega(D^{-7})$ to increase the fitness by a factor of $a^{1/2}$.*

Our final case considers $j^*.f = \times$ with function node j^*-1 active.

Lemma 11. *If nodes D, j^*-1 are active, $j^* > 3$, and $j^*.f = \times$, then a sequence of 4 applications of SAM mutation has a probability of $\Omega(D^{-7})$ to increase the fitness by a factor of $a^{1/3}$.*

The proof of the key lemma combines all the sublemmas as shown in the diagram of Figure 5. The smallest improvement probability $\Omega(D^{-8})$ for $t < 2$, i. e. leading to bound $\mathcal{O}(D^9)$ in Theorem 6, comes from a critical subcase in Lemma 9. That subcase considers that node D is active, node j^* is located after node 3 and is associated with $+$ in the lookup table. However, if j^* receives too low values from its two input connections, i. e. both less than 4, then combining with $t < 2$, a significant fitness improvement requires in the worst case a precise use of nodes 1 and 2 via node j^*, thus probability $\Omega(D^{-8})$. For $t \geq 2$, the setting of Lemma 9 is no longer critical.

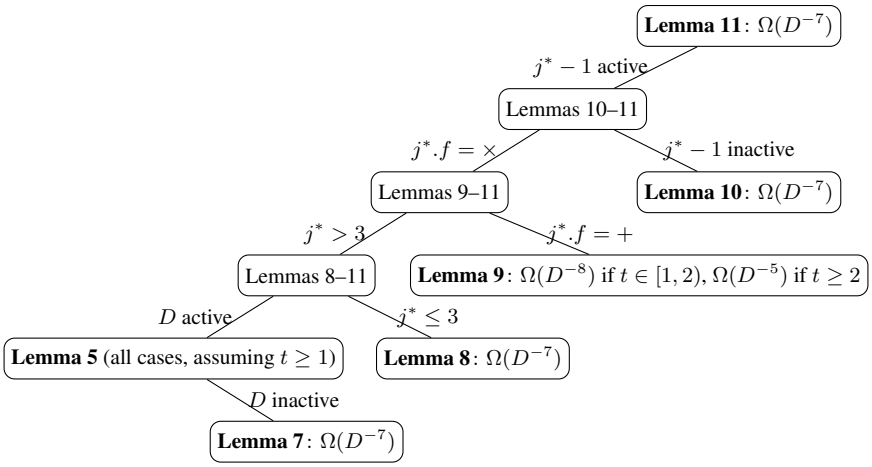


Figure 5: The case analysis constituting the proof of Lemma 5 for $t \geq 1$. From left to right, the proof is branched into smaller sublemmas based on properties of the current program x_0 as described along the branching edges. Lower bounds on the probability of making a significant improvement by a factor of at least $(f(x_0))^{1/3}$ are written next to each sublemma.

Proof of Lemma 5. The result follows from Lemmas 7, 8, 9, 10, and 11 (i. e. see Figure 5). □

7 Experiments

To accompany the theoretical findings, we performed two sets of experiments. In the first, TGP and CGP were compared on MAX with small D to show the exponential performance gain from CGP. In the second, we studied the empirical growth of the average runtime of CGP for larger D.

The setup: Our code is written in Python and we used TinyverseGP [38], a Python library with minimalistic implementations of various flavors of GP, to run the experiments. The $(1+1)$ TGP variants are denoted `simple_tgp_single` and `simple_tgp_multi`, using HVL-prime mutation as described in Section 4 and [39, 40]. The $(1+1)$ CGP variants are `simple_cgp_sam` and `simple_cgp_prob`, using SAM and the probabilistic mutation respectively, as in Algorithms 4 and 3. The code and workflows for reproducing our results are publicly available[2].

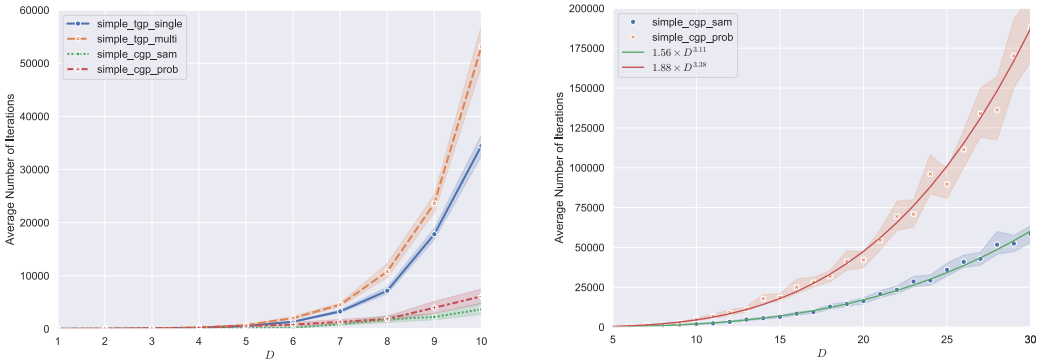
For each experiment, 30 runs were performed with different random seeds and statistics are reported. We also fixed an upper limit 2×10^6 on the number of iterations allowed for each algorithm in each run, however all runs terminated without reaching this. The GP configuration of our experiments is summarized in Table 1. In the first set of experiments, we considered MAX-depth-D-$\{\times, +\}$-$\{1\}$ for $D \in \{1, \ldots, 10\}$ and compared TGP with CGP. In the second set, we tested CGP on MAX-depth-D-$\{\log^\times, \log^+\}$-$\{\log(2)\}$ to support the large output of $D \in \{1, \ldots, 30\}$ as explained below.

MAX param.	**Setting**	**GP param.**	**Setting**
Depth D	$\{1, \ldots, 10\}, \{1, \ldots, 30\}$	Maximum iterations	2×10^6
Input t	$1, 2$	Search & selection strategies	(1+1), non-strict
Function set	$\{\times, +\}$, $\{\log^\times, \log^+\}$	Number of replications per D	30
Terminal set	$\{t\}$, $\{\log t\}$	Mutation rate of prob. mutation	$1/(3D+1)$

Table 1: Configuration of GP and MAX for the experiments.

Log-scaling: A challenge in conducting experiments on MAX-depth-D-$\{\times, +\}$-$\{t\}$ is that the program output can be huge as D grows. At some point, it is necessary to scale down the output and this can be done for $t > 1$ as follows. Here we use $\log$ of base 2 but other bases also work with minor adjustments. We first substitute the input t with $\log t$, then replace the $\mathrm{addition}(a, b)$ in the

[2] `https://github.com/GPTheory/NeurIPS26`



(a) TGP and CGP on MAX-depth-D-$\{+, \times\}$-$\{1\}$.

(b) CGP on MAX-depth-D-$\{\log^+, \log^\times\}$-$\{\log(2)\}$.

Figure 6: Empirical results on MAX problems comparing TGP and CGP for small D (left), and testing CGP for larger D (right). Results for too small D are dropped in the right figure to reduce noise when applying linear regression on the log-log scale to deduce the growth orders.

function table with[3] $\log^+(a,b) := \max(a,b) + \log(1 + 2^{-|a-b|})$ and the $\mathrm{multiplication}(a,b)$ with $\log^\times(a,b) := a + b$. The following lemma shows that these changes will result in a $\log$-transform being applied to the output of every program. The maximum output is then $2^{D-1}\log(\max\{2t, t^2\})$. Since the $\log$ function preserves the order of input values, the (1+1) CGP behaves identically on MAX-depth-D-$\{\log^\times, \log^+\}$-$\{\log t\}$ as on the original problem MAX-depth-D-$\{\times, +\}$-$\{t\}$.

Lemma 12. *Let $f(x)$ be the output value of a CGP program encoded as a vector x of integers and evaluated on* MAX*-depth-D-$\{\times, +\}$-$\{t\}$, and $g(x)$ be the output value of the same encoded program but evaluated on* MAX*-depth-D-$\{\log^\times, \log^+\}$-$\{\log t\}$. It holds that $g(x) = \log(f(x))$.*

The results: Figure 6a shows our results for the first set of experiments, comparing TGP and CGP. It is clearly visible that there is a stark contrast in terms of runtime between TGP and CGP, particularly, the runtime of TGP grows extremely large as D increases. This aligns with our theoretical results and with those of [39, 40]. Additional statistics are provided in the repository.

Figure 6b shows the results for the second set of log-scaling experiments. We observe that CGP is able to obtain solutions for MAX even with large settings of D in a relatively small period of time when compared to the search performance of TGP in Figure 6a. The results suggest that SAM mutation is faster than the probabilistic mutation on MAX. We also deduced the growth orders by applying linear regression to the average runtime in the log-log scale (applying $\log$ to both axes). The observed orders (red and green curves in Figure 6b) are between $\Omega(D^3)$ and $\mathcal{O}(D^4)$. They agree with our theoretical bound $\mathcal{O}(D^8)$ for $t = 2$ but suggest that CGP may run faster in practice.

8 Conclusions

We presented the first rigorous runtime analysis of CGP on the MAX-depth-D-$\{\times, +\}$-$\{t\}$ problem, a well established TGP benchmark whose runtime for mutation-only TGP is known to be superlinear in the optimal program length $n = 2^{D+1} - 1$. We showed that the $(1 + 1)$ CGP algorithm, using either the probabilistic mutation or SAM, finds an optimal program in expected time $\mathcal{O}(\log^9 n)$, yielding an exponential speedup over TGP on this problem. The analysis also identifies how SAM mutation can exploit specific genome locations to obtain large fitness gains. Our theoretical findings are supported by experiments on small depths ($D \leq 10$), which already show a substantial gap between TGP and CGP, and extendable to larger D via log-scaling. Our upper bounds may not be tight as experiments suggest CGP runs faster in practices, but they are sufficient for showing the exponential gap. Our proof requires tedious case analysis, but this was expected for the first step exploring the new search space. We are optimistic that it can be improved in the future. We hope this work provides a useful step towards a better understanding of GP representations for programs.

[3] A more general formula is $\max\{a,b\} - C + \log(2^C + 2^{-|a-b|+C})$ where $C > 1$ can be arbitrarily chosen.

Acknowledgments and Disclosure of Funding

The authors would like to thank Fabricio Olivetti de França who pointed out to us that in an earlier version of this work, we analyzed a more complex form of the MAX problem than the one studied in the literature, and Jakob Bossek for the discussion prior to this work about GP literature. Roman Kalkreuth is supported by an Alexander von Humboldt Professorship in AI held by Holger Hoos.

References

[1] P. Anthes, D. Sobania, and F. Rothlauf. Transformer semantic genetic programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2025*, pages 952–960. Association for Computing Machinery, 2025.

[2] A. Auger and B. Doerr, editors. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, volume 1 of *Series on Theoretical Computer Science*. World Scientific, 2011.

[3] V. D. Bhuva, D.-C. Dang, L. Huber, and D. Sudholt. Evolutionary algorithms for cardinality-constrained ising models. In *Parallel Problem Solving from Nature - PPSN XVII*, pages 456–469. Springer, 2022.

[4] C. Bian, Y. Zhou, M. Li, and C. Qian. Stochastic population update can provably be helpful in multi-objective evolutionary algorithms. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI 2023, pages 5513–5521. IJCAI Organization, 2023.

[5] J. Bossek, F. Neumann, P. Peng, and D. Sudholt. Time complexity analysis of randomized search heuristics for the dynamic graph coloring problem. *Algorithmica*, 83(10):3148–3179, 2021.

[6] K. Cortacero, B. Mckenzie, S. Mueller, R. Khazen, F. Lafouresse, G. Corsaut, N. Acker, F. Frenois, L. Lamant, N. Meyer, B. Vergier, D. G. Wilson, H. Luga, O. Staufer, M. Dustin, S. Valitutti, and S. Cussat-Blanc. Evolutionary design of explainable algorithms for biomedical image segmentation. *Nature Communications*, 14, 11 2023.

[7] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187. Lawrence Erlbaum Associates, 1985.

[8] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[9] D.-C. Dang, A. Eremeev, P. K. Lehre, and X. Qin. Fast non-elitist evolutionary algorithms with power-law ranking selection. In *GECCO '22: Proceedings of the Genetic and Evolutionary Computation Conference*, page 1372–1380. Association for Computing Machinery, 2022.

[10] D.-C. Dang, A. V. Eremeev, and X. Qin. Empirical evaluation of evolutionary algorithms with power-law ranking selection. In *Intelligent Information Processing XII*, page 217–232. Springer Nature, 2024.

[11] D.-C. Dang, A. Neumann, F. Neumann, A. Opris, and D. Sudholt. Theoretical analysis of evolutionary algorithms with quality diversity for a classical path planning problem. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025*, pages 8858–8866. ijcai.org, 2025.

[12] D. Dickmanns, J. Schmidhuber, and A. Winklhofer. Der genetische algorithmus: Eine implementierung in prolog. *Fortgeschrittenenpraktikum, Institut f ur Informatik, Lehrstuhl Prof. Radig, Technische Universit at Munchen*, 1987.

[13] B. Doerr and F. Neumann. A survey on recent progress in the theory of evolutionary algorithms for discrete optimization. *ACM Transactions on Evolutionary Learning and Optimization*, 1 (4), 2021.

[14] B. Doerr and S. Pohl. Run-time analysis of the (1+1) evolutionary algorithm optimizing linear functions over a finite alphabet. In *GECCO '12: Proceedings of the 14th annual conference on Genetic and Evolutionary computation*, page 1317–1324. Association for Computing Machinery, 2012.

[15] B. Doerr, D. Johannsen, and M. Schmidt. Runtime analysis of the (1+1) evolutionary algorithm on strings over finite alphabets. In *FOGA '11: Proceedings of the 11th workshop proceedings on Foundations of Genetic Algorithms*, page 119–126. Association for Computing Machinery, 2011.

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

[17] B. Doerr, D. Johannsen, T. Kötzing, F. Neumann, and M. Theile. More effective crossover operators for the all-pairs shortest path problem. *Theoretical Computer Science*, 471:12–26, 2013.

[18] B. Doerr, C. Doerr, and F. Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104, 2015.

[19] B. Doerr, C. Doerr, and T. Kötzing. Solving problems with unknown solution length at (almost) no extra cost. In *GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, page 831–838. Association for Computing Machinery, 2015.

[20] B. Doerr, T. Kötzing, J. A. G. Lagodzinski, and J. Lengler. Bounding bloat in genetic programming. In *GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference*, page 921–928. Association for Computing Machinery, 2017.

[21] B. Doerr, C. Doerr, and T. Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80(5):1732–1768, May 2018.

[22] B. Doerr, A. Lissovoi, and P. S. Oliveto. Evolving boolean functions with conjunctions and disjunctions via genetic programming. In *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference*, page 1003–1011. Association for Computing Machinery, 2019.

[23] B. Doerr, M. S. Krejca, and N. Vu. Superior genetic algorithms for the target set selection problem based on power-law parameter choices and simple greedy heuristics. In *GECCO '24: Proceedings of the Genetic and Evolutionary Computation Conference*, page 169–177. Association for Computing Machinery, 2024.

[24] B. Doerr, M. S. Krejca, and G. Rudolph. Runtime analysis for multi-objective evolutionary algorithms in unbounded integer spaces. *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence*, 39(25):26955–26963, 2025.

[25] P. Fischer, E. L. Larsen, and C. Witt. First steps towards a runtime analysis of neuroevolution. In *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, FOGA 2023*, pages 61–72. Association for Computing Machinery, 2023.

[26] P. Fischer, J. A. Warwicker, and C. Witt. A runtime analysis of bias-invariant neuroevolution and dynamic fitness evaluation. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2024*. Association for Computing Machinery, 2024.

[27] P. Fiser, J. Schmidt, Z. Vasícek, and L. Sekanina. On logic synthesis of conventionally hard to synthesize circuits using genetic programming. In *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 346–351, 2010.

[28] V. Froese, C. Hertrich, and R. Niedermeier. The computational complexity of ReLU network training parameterized by data dimensionality. *Journal of Artificial Intelligence Research*, 74: 1775–1790, 2022.

[29] C. Gathercole and P. Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. In *Proceedings of the 1st Annual Conference on Genetic Programming*, page 291–296. MIT Press, 1996.

[30] B. W. Goldman and W. F. Punch. Reducing wasted evaluations in cartesian genetic programming. In *EuroGP'13: Proceedings of the 16th European conference on Genetic Programming*, pages 61–72. Springer, 2013.

[31] B. W. Goldman and W. F. Punch. Fast and efficient black box optimization using the parameter-less population pyramid. *Evolutionary Computation*, 23(3):451–479, 2015.

[32] E. Hemberg, S. Moskal, and U. O'Reilly. Evolving code with a large language model. *Genetic Programming Evolvable Machines*, 25(2):21, 2024.

[33] C. Hertrich and M. Skutella. Provably good solutions to the knapsack problem via neural networks of bounded size. *INFORMS Journal on Computing*, 35(5):1079–1097, 2023.

[34] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

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

[36] R. Kalkreuth. *Reconsideration and Extension of Cartesian Genetic Programming*. PhD thesis, Faculty of Computer Science, TU Dortmund University, 2021.

[37] R. Kalkreuth and A. Droschinsky. On the time complexity of simple cartesian genetic programming. In *IJCCI 2019: Proceedings of the 11th International Joint Conference on Computational Intelligen*, pages 172–179. ScitePress, 2019.

[38] R. Kalkreuth, F. O. de França, A. Jankovic, M. Anastacio, J. Dierkes, Z. Vasicek, and H. Hoos. Tinyversegp: Towards a modular cross-domain benchmarking framework for genetic programming. In *GECCO '25 Companion: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 127–130. Association for Computing Machinery, 2025.

[39] T. Kötzing, A. M. Sutton, F. Neumann, and U. M. O'Reilly. The max problem revisited: The importance of mutation in genetic programming. In *GECCO '12: Proceedings of the 14th annual conference on Genetic and evolutionary computation*, page 1333–1340. Association for Computing Machinery, 2012.

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

[41] T. Kötzing, J. A. G. Lagodzinski, J. Lengler, and A. Melnichenko. Destructiveness of lexicographic parsimony pressure and alleviation by a concatenation crossover in genetic programming. *Theoretical Computer Science*, 816(C):96–113, 2020.

[42] J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 1*, pages 768–774. Morgan Kaufmann, 1989.

[43] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex adaptive systems. MIT Press, 1993.

[44] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In *Genetic Programming 1997: Proceedings of the Second International Conference on Genetic Programming*, pages 222–230. Morgan Kaufmann, 1997.

[45] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer Berlin Heidelberg, 2002.

[46] R. T. Lange, Y. Imajuku, and E. Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution, 2025.

[47] J. Lengler and T. Offermann. Diversity-preserving exploitation of crossover. In *FOGA '25: Proceedings of the 18th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 154–165. Association for Computing Machinery, 2025.

[48] M. Li, Z. Weijie, and B. Doerr. Scalable speed-ups for the sms-emoa from a simple aging strategy. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*. IJCAI Organization, 2025.

[49] M. Li, Q. Zhang, W. Zheng, and B. Doerr. Why popular MOEAs are popular: Proven advantages in approximating the pareto front. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[50] A. Lissovoi and P. S. Oliveto. On the time and space complexity of genetic programming for evolving boolean conjunctions. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 2018.

[51] A. Lissovoi and P. S. Oliveto. Computational complexity analysis of genetic programming. In B. Doerr and F. Neumann, editors, *Theory of Evolutionary Computation - Recent Developments in Discrete Optimization*, Natural Computing Series, pages 475–518. Springer, 2020.

[52] Y. Luo, H. Li, Q. Liu, L. Shi, and X. Wu. Node identifiers: Compact, discrete representations for efficient graph learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*. OpenReview.net, 2025.

[53] Z. Lv, C. Bian, C. Qian, and Y. Sun. Runtime analysis of population-based evolutionary neural architecture search for a binary classification problem. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2024*, page 358–366. Association for Computing Machinery, 2024.

[54] Z. Lv, C. Qian, G. G. Yen, and Y. Sun. Analyzing the expected hitting time of evolutionary computation-based neural architecture search algorithms. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 8(6):3899–3911, 2024.

[55] Z. Lv, C. Qian, Y. Liu, J. Fan, and Y. Sun. Runtime analysis of evolutionary NAS for multiclass classification. In *Forty-second International Conference on Machine Learning, ICML 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025.

[56] A. Mambrini and P. S. Oliveto. On the analysis of simple genetic programming for evolving boolean functions. In M. I. Heywood, J. McDermott, M. Castelli, E. Costa, and K. Sim, editors, *Proceedings of the 19th European Conference on Genetic Programming, EuroGP 2016*, pages 99–114. Springer, 2016.

[57] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *GECCO'99: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*, pages 1135–1142. Morgan Kaufmann, 1999.

[58] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Genetic Programming: European Conference, EuroGP 2000*, pages 121–132. Springer Berlin Heidelberg, 2000.

[59] A. Moraglio, K.Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In *Parallel Problem Solving from Nature - PPSN XII*, pages 21–31. Springer, 2012.

[60] F. Neumann, U. M. O'Reilly, and M. Wagner. *Computational Complexity Analysis of Genetic Programming - Initial Results and Future Directions*, pages 113–128. Springer New York, 2011.

[61] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. R. Ruiz, A. Mehrabian, M. P. Kumar, A. See, S. Chaudhuri, G. Holland, A. Davies, S. Nowozin, P. Kohli, and M. Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025.

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

[63] S. Openshaw and I. Turton. Building new spatial interaction models using genetic programming. In *Evolutionary Computing, Lecture Notes in Computer Science*, pages 11–13. Springer Berlin Heidelberg, 1994.

[64] A. Opris. A first runtime analysis of NSGA-III on a many-objective multimodal problem: Provable exponential speedup via stochastic population update. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 8903–8911. International Joint Conferences on Artificial Intelligence Organization, 2025.

[65] A. Opris. Many objective problems where crossover is provably essential. *Artificial Intelligence*, 350:104453, 2026.

[66] T. Perkis. Stack-based genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 148–153. IEEE Press, 1994.

[67] S. Picek, K. Knezevic, D. Jakobovic, and A. Derek. C3po: cipher construction with cartesian genetic programming. In *GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, page 1625–1633. Association for Computing Machinery, 2019.

[68] R. Poli. Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, 1996.

[69] R. Poli and N. F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation*, 11(1):53–66, 2003.

[70] R. Poli and N. F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206, 2003.

[71] R. Poli, N. F. McPhee, and J. E. Rowe. Exact schema theory and markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. *Genetic Programming and Evolvable Machines*, 5(1):31–70, 2004.

[72] C. Qian, Y. Zhang, K. Tang, and X. Yao. On multiset selection with size constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence, 32*. AAAI Press, 2018.

[73] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[74] G. Rudolph. An evolutionary algorithm for integer programming. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation*, page 139–148. Springer Berlin Heidelberg, 1994.

[75] C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming*, pages 83–96, Paris, 1998. Springer-Verlag.

[76] L. Sekanina, J. A. Walker, P. Kaufmann, and M. Platzner. *Evolution of Electronic Circuits*, pages 125–179. Springer Berlin Heidelberg, 2011.

[77] D. Sudholt and C. Thyssen. Running time analysis of ant colony optimization for shortest path problems. *Journal of Discrete Algorithms*, 10:165–180, 2012.

[78] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao. Evolution of deep convolutional neural networks using cartesian genetic programming. *Evolutionary Computation*, 28:141–163, 2020.

[79] L. G. Valiant. A theory of the learnable. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, page 436–445. Association for Computing Machinery, 1984.

[80] L. G. Valiant. Evolvability. *Journal of ACM*, 56(1):3:1–3:21, 2009.

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

[82] D. G. Wilson, S. Cussat-Blanc, H. Luga, and J. F. Miller. Evolving simple programs for playing atari games. In H. E. Aguirre and K. Takadama, editors, *GECCO '18: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 229–236. Association for Computing Machinery, 2018.

[83] D. Wittenberg and F. Rothlauf. Small solutions for real-world symbolic regression using denoising autoencoder genetic programming. In *Genetic Programming - 26th European Conference, EuroGP 2023*, pages 101–116. Springer, 2023.

[84] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR*. OpenReview.net, 2019.

A Proofs for Section 3 (Cartesian Genetic Programming)

Lemma 1. *Let $L := N_{\text{func}}(k+1)+N_{\text{out}}$. Consider a sequence of $k+1$ CGP programs $x_0, \ldots, x_k$ such that for all $i \in [k]$, $f(x_{i-1}) \leq f(x_i)$ and $\text{pmut}'_{\text{SAM}}(x_{i-1}, x_i) \geq p_i > 0$. Then after k iterations the (1+1) CGP using SAM mutation and initialized with program x_0 has probability of at least $\tilde{p} := \prod_{i=1}^{k} p_i$ to change its current program to x_k. Moreover, the (1+1) CGP using the probabilistic mutation with parameter $p = \frac{1}{L}$ under the same condition can reach program x_k with probability at least $\tilde{p} \cdot e^{-k}$, which is $\Omega(\tilde{p})$ if $k = O(1)$. Here the asymptotic notation is w. r. t. L.*

Proof. Since $p_i > 0$ for all $i \in [k]$, all the transitions from x_{i-1} to x_i are feasible for SAM mutation, that is, only one active node is changed per transition. The statement for SAM then holds because the sequence of search points has non-decreasing fitness, thus the target solutions will be accepted once they are created, and moreover the operator is applied independently in k iterations.

For the second statement, let M_i the set of genes where x_{i-1} and x_i differ, we note that $|M_i| \geq 1$. Suppose now that the algorithm starts with solution x_{i-1} and uses the probabilistic mutation instead, then each gene of M_i can be changed with the same probability $p = 1/L$ while the others remain unchanged with probability $(1-1/L)^{L-|M_i|}$. Thus we get

$$\begin{aligned}
&\text{pmut}_{\text{PROB}}(x_{i-1}, x_i) \\
&= \left(1-\frac{1}{L}\right)^{L-|M_i|} \prod_{s \in M_i} \frac{q(s)}{L} = \left(1-\frac{1}{L}\right)^{L-|M_i|} \text{pmut}'_{\text{SAM}}(x_{i-1}, x_i) \\
&\geq \left(1-\frac{1}{L}\right)^{L-1} \text{pmut}'_{\text{SAM}}(x_{i-1}, x_i) \geq \frac{\text{pmut}'_{\text{SAM}}(x_{i-1}, x_i)}{e}.
\end{aligned}$$

Here we use the definition of $\text{pmut}'_{\text{SAM}}(x_{i-1}, x_i)$ as given before the lemma and inequality $(1-1/L)^{L-1} \geq 1/e$. Since this holds for all $i \in [k]$, we deduce that the probabilistic mutation can follow the same path as SAM starting from x_0 to reach x_k with a probability of at least $\tilde{p}/e^k$. The proof is completed by noting that $k = O(1)$. □

The factor $1/e^k$ in the proof is pessimistic because the probabilistic mutation does not need to follow the same path as SAM, e. g. it can be achieved by a direct jump based on the Hamming distance $H(x_0, x_k)$ between the encoded programs x_0 and x_k. However, it is sufficient for our purpose of transferring asymptotic results.

Lemma 2. *For every TGP program x that uses functions from a set F for its internal nodes $I(x)$ and elements of a terminal set T for its leaves, there exists an equivalent CGP program z that produces the same output as x and uses T inputs, F in the lookup table, and at most $|I(x)|$ function nodes. Moreover, if there exist $y_1, y_2 \in I(x)$ such that the subtrees rooted at these nodes are disjoint and identical (reusable subprograms), then z only requires at most $|I(x)| - |I(y_1)|$ function nodes.*

Proof. To build z in CGP, it is clear that we only need to put the functions of F in the lookup table, and to set the elements of T as the inputs. We then label every node $i \in I(x)$ by its depth $d(i)$ in x, and put all the nodes of $I(x)$ as the function nodes of z in a non-increasing order of $d(i)$, i. e. nodes with larger $d(i)$ have smaller indices in z (and ties are broken arbitrarily). The program z is completed by connecting the output to the last function node (with the largest index), and clearly no more than $|I(x)|$ function nodes are used.

Consider that there are disjoint and identical subtrees rooted at y_1 and y_2 in x. Assume w. l. o. g. that $d(y_2) \geq d(y_1)$, then we can delete all nodes of $I(y_1)$ in z and redirect all the severed inputs of nodes s that were connected to a node $r \in I(y_1)$ to the corresponding node $r' \in I(y_2)$. This is possible without re-ordering the nodes in z because $d(s) \leq d(y_1) - 1 < d(y_2)$ thus all nodes s that need to take the output value of y_1 as an argument are located after y_2 in z, and since the subtrees y_1 and y_2 are identical the node r' always exists. Moreover, it follows from the disjointness of the subtrees that the output value of y_2 remains unchanged and equal to that of y_1 after the removal of $I(y_1)$. This reduces the required number of function nodes by $|I(y_1)| = |I(y_2)|$ without changing the output of z. □

B Proofs for Section 4 (The Max Problem)

Lemma 4. *Consider any CGP program for* MAX*-depth-D-$\{\times,+\}$-$\{t\}$ with $t \geq 1$, $D \in \mathbb{N}$. For any function node $i \in [D]$, the domain $\mathcal{V}(i)$ is defined by the following recurrence relation.*

$$\mathcal{V}(i) = \begin{cases} \{t\} & \text{if } i = 0, \\ \{u+v, u \cdot v \mid u, v \in \bigcup_{k=0}^{i-1} \mathcal{V}(k)\} & \text{if } 1 \leq i \leq D. \end{cases}$$

Consequently, it holds that: (i) $\max\{\mathcal{V}(i-1)\} < \max\{\mathcal{V}(i)\}$ *for all* $i \in [D]$*;* (ii) $\min\{\mathcal{V}(i-1)\} \geq t$ *for all* $i \in [D+1]$*;* (iii) *and the maximum output is* $\max\{\mathcal{V}(D)\} = \max\{2t, t^2\}^{2^{D-1}}$.

Proof. The recurrence relation holds because for each pair of (not necessarily distinct) values $u, v \in \bigcup_{k=0}^{i-1} \mathcal{V}(k)$, the function node i offers at most two new values $u+v$ and $u \cdot v$. Statement (i) is also clear because it suffices to set $i.f = +$, $i.\mathrm{in}_1 = i-1$ assuming $\mathrm{val}(i-1)$ has reached $\max\{\mathcal{V}(i-1)\}$ and set $i.\mathrm{in}_2 = 0$. This gives $\mathrm{val}(i) = \mathrm{val}(i.\mathrm{in}_1) + \mathrm{val}(i.\mathrm{in}_2) = \max\{\mathcal{V}(i-1)\} + t > \max\{\mathcal{V}(i-1)\}$, and implies $\max\{\mathcal{V}(i)\} > \max\{\mathcal{V}(i-1)\}$.

Statement (ii) can be shown by induction. The base case of $i = 1$ trivially holds since $\mathcal{V}(0) = \{t\}$. Assume that this holds for all $k \in [i]$, that is, $\min\{\mathcal{V}(k-1)\} \geq t$, we need to show that $\min\{\mathcal{V}(i)\} \geq t$. By the recurrence relation, any $w \in \mathcal{V}(i)$ is either $u+v$ or $u \cdot v$ where $u \in \mathcal{V}(k_u)$ and $v \in \mathcal{V}(k_v)$ for some $k_u \in [i-1]$ and some $k_v \in [i-1]$. It follows from the induction hypothesise that $u \geq t$ and $v \geq t$, and this implies $u+v > t$ and $u \cdot v \geq tv \geq v \geq t$ given that $t \geq 1$. So, all $w \geq t$ and $\min\{\mathcal{V}(i)\} \geq t$.

We show (iii) by induction over D. The base case of $D = 1$ holds because $\mathcal{V}(1) = \{2t, t^2\}$, thus the maximum output is $\max\{2t, t^2\} = \max\{2t, t^2\}^{2^0}$. For all $t \geq 1$, that output is at least $2t \geq 2$, it then follows from (i) that for $D \in \mathbb{N}$, $\max\{\mathcal{V}(i)\} \geq 2$ for all $i \in [D]$. Assume that the maximum output holds for all $D \in [i]$, that is $\max\{\mathcal{V}(i)\} = (\max\{2t, t^2\})^{2^{i-1}} =: w$, we need to show that $\max\{\mathcal{V}(i+1)\} = (\max\{2t, t^2\})^{2^i} = (\max\{2t, t^2\})^{2^{i-1} \cdot 2} = w^2$ with a new function node $i+1$. This w^2-value can be produced by using the node $i+1$ with multiplication and make sure that $(i+1).\mathrm{in}_1 = (i+1).\mathrm{in}_2 = i$ assuming that $\mathrm{val}(i)$ has reached w by the induction hypothesis. It remains to show that w^2 is the largest value of $\mathcal{V}(i+1)$. For all pairs $u, v \in \bigcup_{k=1}^{i} \mathcal{V}(k)$, we get $w^2 = \max\{\mathcal{V}(i)\} \cdot \max\{\mathcal{V}(i)\} \geq u \cdot v$ where the last inequality follows from (i). Moreover, it follows from $w \geq 2$ as shown above that $w^2 \geq 2w = \max\{\mathcal{V}(i)\} + \max\{\mathcal{V}(i)\} \geq u+v$ and here again we use (i). By the definition of $\mathcal{V}(i+1)$ in the recurrent relation, these indeed imply $w^2 = \max\{\mathcal{V}(i+1)\}$. □

Omitted Section (Special Starts and Lower Bound)

We analyze the runtime of the (1+1) CGP with SAM mutation and special initializations. We call a CGP program for MAX-depth-D-$\{\times,+\}$-$\{t\}$ *structurally correct* if the output is connected to the last function node D, and for every function node $i \in [D]$, we have $i.\mathrm{in}_1 = i.\mathrm{in}_2 = i-1$. That it, the program has the same graph structure as the optimal one. Moreover, provided that $t > 1$, no change to the connections is possible without deteriorating the fitness. The structure of the graph is then correct and stable[4]. By a coupon-collecting argument, we can show the following expected time for SAM mutation to correct the function genes, one at a time as all function nodes are active.

Theorem 13. *The expected time for the (1+1) CGP to optimize* MAX*-depth-D-$\{\times,+\}$-$\{t\}$ for any $t > 1$ using SAM mutation and starting from a structurally correct program is $\Theta(D \log D)$.*

Proof. For any $i \in [D]$, it follows from $i.\mathrm{in}_1 = i.\mathrm{in}_2 = i-1$ that $\mathrm{val}(i)$ is either $2\mathrm{val}(i-1)$ or $\mathrm{val}(i-1)^2$. By Lemma 4 (ii), it holds that $\mathrm{val}(i-1) \geq t > 1$, thus in both cases $\mathrm{val}(i) > \mathrm{val}(i-1)$. Thus $\mathrm{val}(i) > \mathrm{val}(i-1)$ for all $i \in [D]$, in other words, the connection genes are optimal and cannot be changed without deteriorating the output value unless the functions selected from the lookup table are changed in parallel. Note that all nodes are active so SAM only allows one change and parallel changes are disallowed.

[4]For $t = 1$, the program can still have the smallest output of 1, thus the structure is not stable and can be easily destroyed.

Therefore, with probability $D/(3D+1) = 1/(3+o(1))$ a function gene is selected and changed. Particularly, changing a function gene that is already optimized, i. e. matches the one in an optimal solution, also leads to the rejection of the offspring. Therefore, we can model the progress using the following coupon collecting process. First, we fix a specific optimal solution, then the coupons are the function genes that are not yet correct with respect to that solution, and in the worst case there can be D of those. In each iteration, with probability $1/(3+o(1))$ we run the standard coupon collector process, that is a coupon arrive, and otherwise, no coupon arrives. It is known that the expected time to collect all coupons in the standard process is DH_D where H_n denotes the n-th harmonic number. Thus the expected runtime of our process is $DH_D/(1/(3+o(1))) = \Theta(D \log D)$. □

We now look at a program where it requires SAM mutation some time to make a change to its current solution. Since this is a suboptimal program, we have a basic lower bound for the expected runtime of the corresponding CGP. We call a CGP program *missing the last* if the output is connected to the function node $D-1$, and for every function node $i \in [D-1]$, we have $i.\mathrm{in}_1 = i.\mathrm{in}_2 = i-1$, and $i.f = \times$ if $i \geq 2$ or $t \geq 2$ and otherwise $i.f = +$, and $D.\mathrm{in}_1 = D.\mathrm{in}_2 = 0$. Figure 3 gives such a program for $D = 4$. To create a new program y of non-decreasing fitness from such a suboptimal solution, it requires within a call to SAM mutation that at least one of $D.\mathrm{in}_1, D.\mathrm{in}_2$ are redirected to node $D-1$, and then that the output is redirected to node D in the last iteration of its while-loop. Summing the probabilities of these events over all possible realizations of SAM mutation gives that the probability to produce y is at most $\mathcal{O}(D^{-4})$. Thus, we obtain the following lower bound.

Theorem 14. *The expected time for the (1+1) CGP to optimize* MAX*-depth-D-$\{\times,+\}$-$\{t\}$ for any $t \geq 1$ using SAM mutation and starting from a missing-the-last program is $\Omega(D^4)$.*

Proof. For a lower bound result, it suffices to look at the waiting time to update the initial program (solution) to a different one. The initial solution is optimal for MAX-depth-$(D-1)$-$\{\times,+\}$-$\{t\}$, thus to obtain a non-deteriorating solution by SAM mutation, the last function node D must be turned active as follows. It requires that at least one of $D.\mathrm{in}_1$ and $D.\mathrm{in}_2$ is changed to $D-1$ (instead of currently 0), and that the unique change to an active node that triggers the exit of the while-loop in SAM connects the output to the last function node D (instead of currently $D-1$). Such an outcome can only be produced if the while-loop runs for $i \geq 2$ iterations in which the first $i-1$ iterations only modify the inactive components $\{D.\mathrm{in}_1, D.\mathrm{in}_2, D.f\}$ and the last correctly modifies the output, thus with probability $\left(\frac{3}{3D+1}\right)^{i-1}\left(\frac{1}{(3D+1)D}\right)$.

Conditioned the above event, we still need at least $D.\mathrm{in}_1$ or $D.\mathrm{in}_2$ to be changed to $D-1$ during the first $i-1$ iterations, and here for an upper bound argument on the probability, we optimistically assume that having any correct changes among $i-1$ is sufficient (even though they may be overwritten by a wrong one later). Note that with probability $2/3$ we change $D.\mathrm{in}_1$ or $D.\mathrm{in}_2$ (and not $D.f$), then NEWVALUE returns $D-1$ with probability of at most $1/(D-1)$ (after excluding the old value). Thus conditioned on the event and by a union bound, the probability of having at least one correct change is at most $(i-1)\cdot\frac{2}{3(D-1)}$. In summary, if the loop runs for $i \geq 2$ iterations, the probability of updating the current solution to a new one is at most $\left(\frac{3}{3D+1}\right)^{i-1}\left(\frac{1}{(3D+1)D}\right)\frac{2(i-1)}{3(D-1)}$. Summing this over all possible i gives that the probability for SAM to update the current solution is at most, here we use $\sum_{k=0}^{\infty} ka^k = a/(1-a)^2$:

$$\begin{aligned}
&\sum_{i=2}^{\infty}\left(\frac{3}{3D+1}\right)^{i-1}\left(\frac{1}{(3D+1)D}\right)\frac{2(i-1)}{3(D-1)}\\
&= \mathcal{O}(D^{-3})\sum_{k=1}^{\infty} k\left(\frac{3}{3D+1}\right)^{k} = \mathcal{O}(D^{-3})\cdot\frac{3/(3D+1)}{(1-3/(3D+1))^2}\\
&= \mathcal{O}(D^{-3})\cdot\frac{3}{3D+1}\cdot\frac{(3D+1)^2}{(3D-2)^2} = \mathcal{O}(D^{-4}).
\end{aligned}$$

Therefore, at least $\Omega(D^4)$ iterations are required in expectation to modify the current solution to a new one. This is also a lower bound on the optimization time as the initial solution is suboptimal. □

C Proofs for Section 5 (General Upper Bound)

Theorem 6. *The (1+1) CGP using either SAM mutation or the probabilistic one with parameter $p = 1/(3D+1)$ and initialized with any program solves* MAX*-depth-D-$\{\times,+\}$-$\{t\}$ for any $t \in [1,2)$ in at most $\mathcal{O}(D^9)$ iterations, and this bound is dropped to $\mathcal{O}(D^8)$ for $t \geq 2$.*

Proof. We first show the result for SAM mutation. The run of the algorithm is divided into two phases: the first ends when the output of the current program is at least $\max\{2t, t^2\}$ and the second phase is completed when the optimal output is obtained. We first consider the case that $t \in [1,2)$.

To complete the first phase, it suffices to make sure that the first function node is optimized, i. e. reaching its $\max\{\mathcal{V}(1)\} = \max\{2t, t^2\}$, and active. Once this value is reached it is propagated, as the node is active, to the output either directly, or by multiplying or adding some other values of at least $t \geq 1$ by Lemma 4 (ii) along the way. This guarantees that the output is at least $\max\{2t, t^2\}$, and can be achieved in at most 2 consecutive calls to SAM mutation as follows. The operator first corrects $1.f$ if required (probability at least $1/(3D+1)$), then if required points the output to the first function node (probability at least $1/((3D+1)D)$). Note the first operation does not decrease the fitness and hence, it is feasible. The first phase therefore has a probability of $\Omega(1/D^3)$ to be completed per two iterations, thus it is done after $\mathcal{O}(D^3)$ iterations in expectation.

In the second phase, we apply Lemma 5 to estimate the expected length of the phase. We assume that by the end of the first phase, the fitness of the current solution x_0 is at least $a := f(x_0) \geq \max\{2t, t^2\}$, and let $M := \max\{2t, t^2\}^{2^{D-1}}$ be the maximum output as shown in Lemma 4 (iii). It follows from Lemma 5 that after at most expected $\mathcal{O}(\ell \cdot D^8)$ iterations, the fitness is increased to at least $\min\{a^{(4/3)^\ell}, M\}$. By setting $\ell := \lceil \frac{D-1}{\log(4/3)} \rceil = O(D)$, we get

$$(4/3)^\ell \geq (4/3)^{\frac{D-1}{\log(4/3)}} = \left((4/3)^{\log_{4/3} 2}\right)^{D-1} = 2^{D-1}.$$

In other words, after at most $\mathcal{O}(\ell \cdot D^8) = \mathcal{O}(D^9)$ iterations, the optimal output $\min\{a^{(4/3)^\ell}, M\} = M$ is reached.

Combining the two phases gives that the (1+1) CGP using SAM mutation produces the optimal program after at most $\mathcal{O}(D^3) + \mathcal{O}(D^9)$ iterations. The bound for $t \geq 2$ follows by the same proof except the expected length of the second phase, which dominates the runtime, is shortened to $\mathcal{O}(D^8)$. The result for the probabilistic mutation follows from Lemma 1 since we only use probability estimates from $\mathrm{pmut}'_{\mathrm{SAM}}$ and the mutation parameter is $p = 1/(3D+1) = 1/L$. □

D Proofs for Section 6 (Time to a Significant Improvement)

Lemma 7. *If the function node D is inactive then a single application of SAM mutation has a probability of at least $\Omega(D^{-7})$ to increase the fitness by a factor of a.*

Proof. Let $k < D$ be the node that the output is currently pointed to, i. e. $\mathrm{val}(k) = a$. SAM only needs to make the following changes to the inactive components (if required): setting $D.\mathrm{in}_1 = D.\mathrm{in}_2 = k$ (probability $1/((3D+1)(D-1))^2$), and $D.f = \times$ (probability $1/(3D+1)$). Finally, it points the output to node D (probability $1/((3D+1)D)$) and this terminates the while-loop. The fitness of the new solution is then a^2, and the probability of the event per call to SAM is $\Omega(D^{-7})$. □

Lemma 8. *If function node D is active and $j^* \leq 3$ then a sequence of 7 applications of SAM mutation has a probability of at least $\Omega(D^{-7})$ to optimize* MAX*-depth-D-$\{\times,+\}$-$\{t\}$.*

Proof. We consider the worst case of $j^* = 3$ then the other cases follow as less work is required for SAM (by skipping (ii)). The operator can make the following changes (if required) in order: (i) setting $2.\mathrm{in}_1 = 2.\mathrm{in}_2 = 1$ (probability $1/(3D+1)^2$); (ii) setting $3.\mathrm{in}_1 = 3.\mathrm{in}_2 = 2$ (probability $1/(2(3D+1))^2$); (iii) correcting $1.f$, then $2.f$ and finally $3.f$ (probability $1/(3D+1)$ per correction). These steps (at most 7) carrying out in that order do not create a program with deteriorating fitness, hence possible immediate programs created (as we do not know the active status of function nodes $1, 2$ and 3), are always accepted by the algorithm. We conclude that after at most 7 calls to SAM, the probability to directly optimize output is at least $\Omega(D^{-7})$. □

Lemma 10. *If function node D is active but j^*-1 is not, and $j^* > 3$ and $j^*.f = \times$, then a single application of SAM mutation has a probability of $\Omega(D^{-7})$ to increase the fitness by a factor of $a^{1/2}$.*

Proof. The following changes in order (if required) are sufficient for SAM: (i) setting $(j^*-1).\mathrm{in}_1 = (j^*-1).\mathrm{in}_2 = n_v$ (probability at least $1/((3D+1)(D-2))^2$); (ii) turning $(j^*-1).f$ to $\times$ (probability $1/(3D+1)$); (iii) finally setting $j^*.\mathrm{in}_1 = j^*-1$ (probability $1/((3D+1)(D-1))$). Note that only the last change is for an active node, thus one call to SAM is sufficient with a success probability of $\Omega(D^{-7})$. If successful, the fitness is increased by a factor of

$$\frac{((v^2)\cdot v)^{2^{D-j^*}}}{a} \geq \frac{(v^3)^{2^{D-j^*}}}{(v^2)^{2^{D-j^*}}} = v^{2^{D-j^*}} = \left((v^2)^{2^{D-j^*}}\right)^{1/2},$$

which is at least $a^{1/2}$. □

Lemma 11. *If nodes D, j^*-1 are active, $j^* > 3$, and $j^*.f = \times$, then a sequence of 4 applications of SAM mutation has a probability of $\Omega(D^{-7})$ to increase the fitness by a factor of $a^{1/3}$.*

Proof. It holds that exactly one input of j^* must be pointed to node j^*-1. This is firstly because by the definition of j^*, all nodes $\ell > j^*$ have used up all their inputs to the immediate preceding nodes, thus only the inputs of j^* are available to make j^*-1 active. Secondly, it is not possible to have both inputs of j^* pointing to j^*-1 because otherwise that contradicts the definition j^*.

In the case that $\mathrm{val}(j^*-1)$ provides the input the smaller value for j^*, i. e. $n_u = j^*-1 \wedge n_v < j^*-1$, it is sufficient for SAM to re-create the same configuration of the solution as in the proof of the previous lemma. However, it requires less work but spans across multiple calls: (i) setting $(j^*-1).\mathrm{in}_1 = (j^*-1).\mathrm{in}_2 = n_v$ (probability at least $1/((3D+1)(D-2))^2$), and this will result in a non-deteriorating solution because $\mathrm{val}((j^*-1).\mathrm{in}_1), \mathrm{val}((j^*-1).\mathrm{in}_2) \leq u \leq v$; (ii) finally turning $(j^*-1).f$ to $\times$ (probability $1/(3D+1)$) which does also not decrease the fitness. Thus the success probability is $\Omega(D^{-5})$ after at most 3 calls to SAM, and if successful, by the same argument as in the previous lemma, the fitness is increased by a factor of $a^{3/4}$.

Otherwise if $\mathrm{val}(j^*-1)$ is the input of the larger value of j^*, i. e. $n_u < j^*-1 \wedge n_v = j^*-1$, we consider the two following subcases.

If $v \geq u^2$: It is then sufficient for SAM to redirect $j^*.\mathrm{in}_1$ to j^*-1, thus only one call to the operator has a success probability of $1/((3D+1)(D-1)) = \Omega(D^{-2})$. This is because the fitness is then increased by a factor of

$$\frac{(v^2)^{2^{D-j^*}}}{(uv)^{2^{D-j^*}}} = \frac{(v^{2/3}\cdot v^{4/3})^{2^{D-j^*}}}{(uv)^{2^{D-j^*}}} \geq \frac{((u^2)^{2/3}\cdot v^{4/3})^{2^{D-j^*}}}{(uv)^{2^{D-j^*}}} = a^{1/3}.$$

If $v < u^2$: The following changes can be done (if necessary): (i) redirecting $(j^*-1).\mathrm{in}_1 = (j^*-1).\mathrm{in}_2 = n_u$ (probability at least $1/((3D+1)(D-2))^2$) which is feasible since we will only make this change if the incoming value is smaller than u; (ii) turning $(j^*-1.f)$ to $\times$ (probability $1/(3D+1)$) which is also feasible due to $u \geq 4$; (iii) redirecting $j^*.\mathrm{in}_1 = j^*-1$ (probability $1/((3D+1)(D-1))$) which is also feasible, since $u^4 \geq u^3$. So all these steps in order do not decrease the fitness and hence, are feasible for SAM. Thus at most 4 calls to SAM are required and the success probability is $\Omega(D^{-7})$ with an increase in fitness by a factor of at least

$$\frac{(u^4)^{2^{D-j^*}}}{(uv)^{2^{D-j^*}}} = \frac{(u^{4/3}\cdot (u^2)^{4/3})^{2^{D-j^*}}}{(uv)^{2^{D-j^*}}} \geq \frac{(u^{4/3}\cdot v^{4/3})^{2^{D-j^*}}}{(uv)^{2^{D-j^*}}} = a^{1/3},$$

and this concludes the proof. □

E Proofs for Section 7 (Experiments)

Lemma 12. *Let $f(x)$ be the output value of a CGP program encoded as a vector x of integers and evaluated on* MAX*-depth-D-$\{\times,+\}$-$\{t\}$, and $g(x)$ be the output value of the same encoded program but evaluated on* MAX*-depth-D-$\{\log^\times, \log^+\}$-$\{\log t\}$. It holds that $g(x) = \log(f(x))$.*

Proof. We will prove a more general result that the value, denoted $\mathrm{val}_g(i)$, of any node indexed i in the phenotype graph on MAX-depth-D-$\{\log^\times, \log^+\}$-$\{\log t\}$, regardless whether this node is active or not, is the $\log$-transform of the value, denoted $\mathrm{val}_f(i)$, of the same node on the original problem MAX-depth-D-$\{\times, +\}$-$\{t\}$. That is, $\mathrm{val}_g(i) = \log(\mathrm{val}_f(i))$. The lemma then follows because the program output is connected to one of these nodes and returns its value.

The proof is done by induction on i. The base case $i = 0$ (the input node) holds by the definition of the terminal set in the transformed problem MAX-depth-D-$\{\log^\times, \log^+\}$-$\{\log t\}$, that is, $\mathrm{val}_g(0) = \log t = \log(\mathrm{val}_f(0))$. In the induction step, we need to prove the result for (function) node indexed $i + 1$ where $i \geq 0$, that is $\mathrm{val}_g(i + 1) = \log(\mathrm{val}_f(i + 1))$, assuming that this holds for all indices $j \leq i$. Let a, b be the indices of the node(s) that the two input connections of node $i+1$ are currently connected to. By the design of CGP programs, we get $a \leq i$ and $b \leq i$, then it holds by the induction hypothesis (I.H.) that $\mathrm{val}_g(a) = \log(\mathrm{val}_f(a))$ and $\mathrm{val}_g(b) = \log(\mathrm{val}_f(b))$.

In the case that node $i+1$ is associated with $\times$ in the lookup table of the original problem, and hence $\log^\times$ in MAX-depth-D-$\{\log^\times, \log^+\}$-$\{\log t\}$, we have:

$$\begin{aligned}\mathrm{val}_g(i+1) &= \log^\times(\mathrm{val}_g(a), \mathrm{val}_g(b)) \overset{\text{def}}{=} \mathrm{val}_g(a) + \mathrm{val}_g(b) \overset{\text{I.H.}}{=} \log(\mathrm{val}_f(a)) + \log(\mathrm{val}_f(b)) \\ &= \log(\mathrm{val}_f(a) \times \mathrm{val}_f(b)) = \log(\mathrm{val}_f(i+1)).\end{aligned}$$

Otherwise if node $i + 1$ is associated with $+$ in the lookup table of the original problem and so $\log^+$ in the transformed one, and assuming w. l. o. g. that $\mathrm{val}_g(a) \geq \mathrm{val}_g(b)$, we get:

$$\begin{aligned}\mathrm{val}_g(i+1) &= \log^+(\mathrm{val}_g(a), \mathrm{val}_g(b)) \overset{\text{def}}{=} \max\{\mathrm{val}_g(a), \mathrm{val}_g(b)\} + \log(1 + 2^{-|\mathrm{val}_g(a) - \mathrm{val}_g(b)|}) \\ &= \mathrm{val}_g(a) + \log(1 + 2^{-(\mathrm{val}_g(a) - \mathrm{val}_g(b))}) = \log\left(2^{\mathrm{val}_g(a)}\left(1 + 2^{-(\mathrm{val}_g(a) - \mathrm{val}_g(b))}\right)\right) \\ &= \log\left(2^{\mathrm{val}_g(a)} + 2^{\mathrm{val}_g(b)}\right) \overset{\text{I.H.}}{=} \log\left(2^{\log(\mathrm{val}_f(a))} + 2^{\log(\mathrm{val}_f(b))}\right) \\ &= \log(\mathrm{val}_f(a) + \mathrm{val}_f(b)) = \log(\mathrm{val}_f(i+1)).\end{aligned}$$

Combining the two cases completes the induction step and hence the proof. □

Although $\log$-transform was not needed in our experiments with TGP, we note that the statement of the lemma also holds for TGP programs. The proof of this follows the same steps as in the above proof, but the induction is done on the height of TGP subprograms.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: Our abstract and introduction claim a proof for the performance guarantees of Cartesian Genetic Programming (CGP) on the MAX problems, showing an exponential speed-up compared to Tree-based GP (TGP). The result sheds the light on the benefit of the graph-based representation that CGP brings and our analysis shows how CGP can navigate its search and fitness spaces efficiently.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: We discussed the limitations of our results in the conclusions. While our upper bounds may not be tight, as experiments suggest CGP may run faster in practice, they are sufficient for demonstrating the exponential speedup. Note that experiments can only explore limited values for D, while the theoretical result holds asymptotically as D grows to infinity. Our proof requires tedious case separation, a necessary step for establishing a first result in a new territory. Despite these challenges, we remain optimistic that these can be improved upon in future work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes].

   Justification: Our main result (Theorem 6) is clearly stated with all necessary assumptions. The algorithms, operators and benchmark function it covers are precisely defined in Sections 3 and 4. The necessary lemmas constructing its proof are stated in the paper, and their proofs along the proof of the main result are fully given in the appendix document. Due to space restriction, some results non-essential for our message are deferred to the appendix and their proofs are given there.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have described the experimental setup in detail, disclosed the source code that was used for the experiments and provided the plot scripts to reproduce the figures.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code used for the experiments builds upon a open source benchmarking tool. The link to the modified, anonymized version that has been used for this specific use case, is provided in the paper. The execution scripts are also part of the content of the repository. The benchmarking tool used is properly documented and test scripts are provided to exemplify its usage.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The problem considered does not require a training and test data split. However, we have listed the settings of the hyperparameters. Please note, since we only considered simple models for our experiments used to complement theoretical findings, we did not perform hyperparameter optimization.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Confidence intervals (CIs) are included in our plots, and additional statistics are provided in the appendix document. Our empirical results indicate a clear performance gap between CGP and TGP, and there are no overlaps of CIs for D that we are interested in, so significance tests are not required. We emphasize that our main contribution is theoretical for the runtime of CGP in expectation, and the experiments were conducted to only complement the theoretical findings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The general experimental setup is described in detail in the experiments section. The source code is publicly available and the repository also contains the execution scripts. In addition we report technical setup that was used to perform the experiments:

   - CPU Partition of our local (non-public) compute cluster
     - 18 CPU Nodes with 2x AMD EPYC 7543 32-Core Processor, the first four nodes have 32x64GB of RAM, the rest 16x64GB
     - 30 single-core jobs with 3GB of maximum memory capacity ran in parallel per setting of D
     - Maximum runtime for each job was limited to 24 hours. The maximum runtime for each experiment was therefore also 24 hours.
     - We performed a round of preliminary experiments, with a lower sample size to test the implementations and general setup.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: We have read and strictly follow the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A].

    Justification: Our work is purely theoretical, there are no foreseeable negative societal impacts as we know of. The main contribution of our paper is to advance our understanding of search spaces of programs, beyond the conventional tree-based representation, in GP.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A].

Justification: The paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes].

Justification: We used the TinyverseGP library for our experiments, cited the original paper, and provided the required license in our public repository for our code and experiments.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A].

Justification: The paper does not release new assets. We rigorously analyzed an existing search paradigm in GP.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.