Theoretical Computer Science 816 (2020) 96–113

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Destructiveness of lexicographic parsimony pressure and alleviation by a concatenation crossover in genetic programming

Timo Kötzing[a], J.A. Gregor Lagodzinski[a], Johannes Lengler[b], Anna Melnichenko[a]

[a] Hasso Plattner Institute, University of Potsdam, Germany
[b] ETH Zürich, Switzerland

A R T I C L E   I N F O

Article history:
Received 18 January 2019
Received in revised form 24 November 2019
Accepted 27 November 2019
Available online 3 December 2019
Communicated by B. Doerr

Keywords:
Genetic programming
Mutation
Theory
Run time analysis

A B S T R A C T

For theoretical analyses there are two specifics distinguishing GP from many other areas of evolutionary computation: the variable size representations, in particular yielding a possible bloat (i.e. the growth of individuals with redundant parts); and also the role and the realization of crossover, which is particularly central in GP due to the tree-based representation. Whereas some theoretical work on GP has studied the effects of bloat, crossover had surprisingly little share in this work.
We analyze a simple crossover operator in combination with randomized local search, where a preference for small solutions minimizes bloat (lexicographic parsimony pressure); we denote the resulting algorithm Concatenation Crossover GP. We consider three variants of the well-studied Majority test function, adding large plateaus in different ways to the fitness landscape and thus giving a test bed for analyzing the interplay of variation operators and bloat control mechanisms in a setting with local optima. We show that the Concatenation Crossover GP can efficiently optimize these test functions, while local search cannot be efficient for all three variants independent of employing bloat control.

1. Introduction

Genetic Programming (GP) is a field of Evolutionary Computing (EC) where the evolved objects encode programs. Usually a tree-based representation of a program is iteratively improved by applying variation operators (mutation and crossover) and selection of suitable offspring according to their quality (fitness). Most other areas of EC deal with fixed-length representations, whereas the tree-based representation distinguishes GP. This representation of variable size leads to one of the main problems when applying GP: *bloat*, which describes an unnecessary growth of representations. Solutions may have many redundant parts, which could be removed without affecting the quality, and search is slowed down, wasted on uninteresting areas of the search space.

In this paper we study GP from the point of view of run time analysis. While many previous theoretical works analyzed mutational GP with the offspring produced by varying a single parent, we analyze a GP algorithm employing a simple crossover with the offspring produced from two parents. Although our crossover is far from practical applications of GP (it merely concatenates the two parent trees), this simple setting aims at understanding the interplay between (our variant of)

E-mail address: johannes.lengler@inf.ethz.ch (J. Lengler).

https://doi.org/10.1016/j.tcs.2019.11.036
0304-3975/© 2019 Elsevier B.V. All rights reserved.

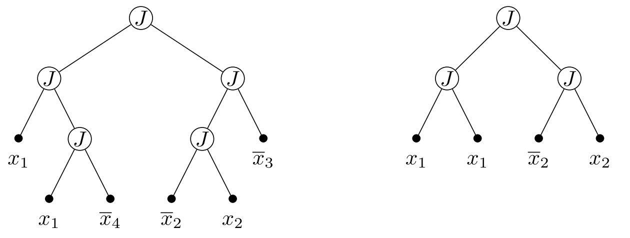


Fig. 1. Two GP-trees with the same fitness. For 2/3-Majority the fitness is 2 and for $+c$-Majority with $c = 2$ the fitness is 1. However, the left one has size 6 whereas the right one has size 4.

crossover, the problem of bloat and *lexicographic parsimony pressure*, a method for bloat control introduced in [21]. Other theoretical work in GP has analyzed different problems and phenomena, in particular for the Probably Approximately Correct (PAC) learning framework [15], the Max-Problem [8,19,16] as well as Boolean functions [4,20,23,24,28].

For the effects of bloat in the sense of redundant parts in the tree, we draw on previous theoretical works that analyzed this phenomenon, especially [29] and [3]. In these, the fitness function Majority as introduced in [9] was analyzed. This fitness function aims at providing a simple test function for studying GP algorithms: it gives deterministic fitness evaluations (rather than those based on estimating the quality of the GP-tree by sampling different inputs for the GP-tree understood as a function) and the tree structure is not relevant for the fitness (making the function amenable for theoretical analyzes). A further theoretical work discussing bloat in GP is [35].

Individuals for Majority are binary trees, where each inner node is labeled J (short for *join*, but without any associated semantics) and leaves are labeled with variable symbols; we call such trees *GP-trees*. The set of variable symbols is $\{x_1, \dots, x_n\} \cup \{\overline{x}_1, \dots, \overline{x}_n\}$, for some n. In particular, variable symbols are paired: x_i is paired with $\overline{x}_i$. For Majority, we call a variable symbol x_i *expressed* if there is a leaf labeled x_i and there are at least as many leaves labeled x_i as there are leaves labeled $\overline{x}_i$; the positive instances are in the majority. The fitness of a GP-tree is the number of its expressed variable symbols x_i. See Fig. 1 for two exemplary GP-trees. This setting captures two important aspects of GP: variable length representations and that any given functionality can be achieved by many different representations, i.e., multiple very different trees can have the same set of variables expressed. However, the tree-structure, typically crucial in GP problems, is completely unimportant for the Majority function.

We know that Majority can be efficiently optimized by a mutational GP called (1+1) GP (see Algorithm 1 for details, basically performing a randomized local search). This holds in the case preferring shorter representations by lexicographic parsimony pressure, as shown in [29], as well as in the case without such preference [3]. Similar to recent literature on theory of GP, we will consider lexicographic parsimony pressure as our method of bloat control and henceforth only speak of *bloat control* to denote this method. We note, however, that the GP literature knows many more methods for controlling bloat [22]; for example, the depth of the GP-tree can be limited, either as a hard constraint or a soft constraint.

In addition to weighted versions of Majority, another, similar fitness function Order (see also [6,30]) has been considered, but neither of these provide us with strong differences in the optimization behavior of different GP algorithms. Thus, we propose three variants of Majority, called $+c$-Majority, 2/3-Majority and 2/3-SuperMajority, which negatively affect the optimization of certain GP algorithms.

For $+c$-Majority a variable is expressed if its positive literals are not only in the majority, but also there has to be at least c more positive than negative literals. On the one hand, we show that a random GP-tree with a linear number of leaves expresses any given variable with constant probability. On the other hand, with constant probability such a tree has a majority of negative literals of any given variable (indeed, there is a constant probability that the variable has neither positive nor negative literals in the GP-tree). This yields a plateau of equal fitness which can only be overcome by adding c positive literals, i.e., we need a rich set of neutral mutations that allow genetic drift to happen. Bloat control suppresses this genetic drift by biasing the search towards smaller solutions. Specifically, it may not allow to add positive literals one by one, which results in an infinite run time (see Lemma 3.1). Note that allowing the local search to add c leaves at the same time still results only in a small chance of $O(n^{-c})$ of jumping the plateau. Hence, the $+c$-Majority fitness function serves as an example where bloat control explicitly harms the search.

For 2/3-Majority, a variable is expressed if its positive literals hold a 2/3 majority, i.e., if 2/3 of all its literals are positive. The fitness associated with 2/3-Majority is the number of expressed variables while for 2/3-SuperMajority each expressed variable contributes a score between 1 and 2, where larger majorities give larger scores (see Section 2 for details). The variant 2/3-SuperMajority is utilized to aggravate the effect of bloat since it rewards large numbers of (positive) literals. We show that local search with bloat control is efficient for these two problems (Theorems 3.3 and 3.6). However, without bloat control local search fails on 2/3-SuperMajority due to bloat (see Theorem 4.1). Note that some of the theorems are given only for certain ranges of initial tree sizes, but the reasonable size of $n/2$ is always included.

Regarding optimization without bloat control, we obtain experimental results as depicted in Fig. 4. They provide a strong indicator that, when no bloat control is applied, optimization of $+c$-Majority is efficient, in contrast to the case of bloat control. The trend for 2/3-Majority indicates that optimization proceeds significantly more slowly without bloat control than with bloat control. Nevertheless, optimization seems to be feasible in contrast to the case of 2/3-SuperMajority.

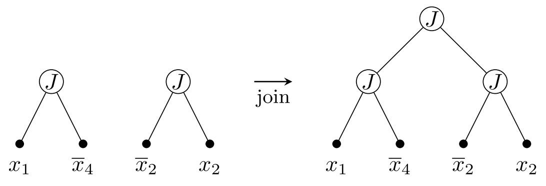


Fig. 2. Two GP-trees are joined as in Algorithm 2. A new join-node is introduced and connected to both trees.

Table 1
Overview of the results of the paper. A check mark denotes optimization in polynomial time with high probability, a cross denotes superpolynomial optimization time. A check mark with a subscript e denotes the results obtained experimentally. Note that the cited theorems are conditional on certain initial tree sizes.

Problem class	Local search		Crossover
	w/bloat control	w/o bloat control	w/bloat control
$+c$-Majority	$\times$ Theorem 3.1	$\checkmark_e$ Fig. 4	$\checkmark$Theorem 5.1
2/3-Majority	$\checkmark$Theorem 3.3	$\checkmark_e$ Fig. 4	$\checkmark$Theorem 5.1
2/3-SuperMajority	$\checkmark$Theorem 3.6	$\times$ Theorem 4.1	$\checkmark$Theorem 5.5

Subsequently, we study a simple crossover which works as follows. The algorithm maintains a population of λ individuals, which are initialized randomly before a local search with bloat control is performed for a number of iterations. As a local search we employ the (1+1) GP, a simple mutation-only GP which iteratively either adds, deletes, or substitutes a vertex of the tree. We employ this algorithm for a number of rounds large enough to ensure that each vertex of the tree has been considered for deletion at least once with high probability, which aims at controlling bloat. Afterwards, the optimization proceeds in rounds; in each round, each individual t_0 is mated with a random other individual t_1 by joining t_0 and t_1 to obtain a tree t' which contains both t_0 and t_1 (see Fig. 2); then local search is performed on t' as before yielding a tree t''. If t'' is at least as fit as t_0, we replace t_0 in the population by t''. The algorithm is called *Concatenation* since it joins two individuals, which is basically a concatenation. This algorithm is similar in spirit to other GP crossover operators from the literature, for example geometric semantic crossover as discussed in [27]: the parent individuals are copied verbatim into the offspring, which carries further tree nodes to connect the parents in a useful way. Note that our crossover operator is different from other approaches for memetic crossover GP as found, for example, in [7]; it is also very different from many GP crossovers found in the literature because of its almost complete disregard for the tree structure of the individuals. However, this crossover already highlights some benefits which can be obtained with crossover, and it has the great advantage of being analyzable. We do not suggest that this crossover should be applied in practice; it serves the purpose of furthering theoretical analyzes in the area of GP.

We show that the Concatenation Crossover GP with bloat control efficiently optimizes all three test functions $+c$-Majority, 2/3-Majority as well as 2/3-SuperMajority, due to its ability to combine good solutions (see Theorem 5.1). It crucially makes use of bloat control when using local search in order to remove unnecessary parts of the search tree, of which there is a considerable amount due to the plain concatenation of two solutions. The diversity is maintained by never truly discarding any of the initially produced individuals, but only considering them for replacement by their offspring which include their parents as part of the concatenation operation of the crossover. We summarize our findings in Table 1.

In Section 2 we state the formal definitions of algorithms and problems, as well as the mathematical tools we use. Section 3 gives the results for local search *with* bloat control, Section 4 for local search *without* bloat control and Section 5 for the Concatenation Crossover GP. In Section 6 we show and discuss our experimental results, before Section 7 concludes the paper.

2. Preliminaries

For a given n we let $[n] = \{1, \ldots, n\}$ be the set of variables. The only non-terminal (function symbol) is J of arity 2; the *terminal set* X consists of $2n$ literals, where $\overline{x}_i$ is the complement of x_i:

$$F := \{J\},\ J \text{ has arity } 2, \qquad X := \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}.$$

For a GP-tree t, we denote by $S(t)$ the set of leaves in t. By $S_i^+(t)$ and $S_i^-(t)$ we denote the set of leaves that are x_i-literals and $\overline{x}_i$-literals, respectively, and by $S_i(t) := S_i^+(t) \cup S_i^-(t)$ we denote the set of all i-literals. By $S^+(t) := \bigcup_{i=1}^n S_i^+(t)$ and $S^-(t) := \bigcup_{i=1}^n S_i^-(t)$ we denote the set of all positive and negative leaves, respectively. We denote the sizes of all these sets by the corresponding lower case letters, i.e., $s(t) := |S(t)|$, $s_i(t) := |S_i(t)|$, etc. In particular, we refer to $s(t)$ as the *size* of t.

On the syntax trees, we analyze the problems $+c$-Majority, 2/3-Majority, and 2/3-SuperMajority, which are defined as

Given a GP-tree t, mutate t by applying HVL-Prime. For each application, choose uniformly at random one of the following three options.

substitute	Choose a leaf uniformly at random and substitute it with a leaf in X selected uniformly at random.
insert	Choose a node $v \in X$ and a leaf $u \in t$ uniformly at random. Substitute u with a join node J, whose children are u and v, with the order of the children chosen uniformly at random.
delete	Choose a leaf $u \in t$ uniformly at random. Let v be the sibling of u. Delete u and v and substitute their parent J by v.

Fig. 3. Mutation operator HVL-Prime.

$$\begin{aligned}
+c\text{-}\textsc{Majority} &:= |\{i \in [n] \mid s_i^+ \geq s_i^- + c\}| \,; \\
2/3\text{-}\textsc{Majority} &:= |\{i \in [n] \mid s_i \geq 1 \text{ and } s_i^+ \geq \tfrac{2}{3} s_i\}| \,; \\
2/3\text{-}\textsc{SuperMajority} &:= \sum_{i=1}^{n} f_i, \text{ where} \\
f_i &:= \begin{cases} 0 & \text{, if } s_i = 0 \text{ or } s_i^+ < \tfrac{2}{3} s_i, \\ 2 - 2^{s_i^- - s_i^+} & \text{, otherwise.} \end{cases}
\end{aligned}$$

We call a variable contributing to the fitness *expressed*. Since both $+c$-Majority and 2/3-Majority count the number of expressed variables, they take values between 0 and n. The function 2/3-SuperMajority is similar to 2/3-Majority, but if a 2/3 majority is reached 2/3-SuperMajority awards a bonus for larger majorities: the term f_i grows with the difference $s_i^+ - s_i^-$. Since $f_i \leq 2$, the function 2/3-SuperMajority takes values in $[0, 2n]$. Note that the value $2n$ can never actually be reached, but can be arbitrarily well approximated.

In this paper we consider simple mutation-based genetic programming algorithms which use a modified version of the Hierarchical Variable Length (HVL) operator ([33], [34]) called HVL-Prime as discussed in [6]. HVL-Prime allows trees of variable length to be produced by applying three different operations: insert, delete and substitute (see Fig. 3). Each application of HVL-Prime chooses one of these three operations uniformly at random. We note that the literature also contains variants of the mutation operator that apply several such operations simultaneously (see [6,30]).

The first algorithm we study is the (1+1) GP. The algorithm is initialized with a tree generated by s_{init} random insertions. Afterwards, it maintains the best-so-far individual t. In each round, it creates an offspring of t by mutation. This offspring is discarded if its fitness is worse than t, otherwise it replaces t. We recall that the fitness in the case with bloat control contains the size as a second order term. Algorithm 1 states the (1+1) GP more formally.

Algorithm 1: (1+1) GP with mutations according to Fig. 3.

```
1 Let $t$ be a random initial tree of size $s_{\text{init}}$;
2 while optimum not reached do
3     $t' \leftarrow \text{mutate}(t)$;
4     if $f(t') \geq f(t)$ then $t \leftarrow t'$
```

2.1. Crossover

The second algorithm we consider is population-based. When introduced by J. R. Koza [17], Genetic Programming used fitness-proportionate selection and a genetic crossover, however mutation was hardly considered. In subsequent works many different setups for the crossover operator were introduced and studied. For instance, in [33] combinations of GP with local search in the form of mutation operators were studied and yielded better performance than GP.

Usually, two parents (a *current solution* and a *mate*) are used to generate a number of offspring. These offspring are a recombination of the alleles from both parents derived in a probabilistic manner. By modeling each individual as a GP-tree, a crossover-point in both parents is decided upon due to a heuristic and the subtrees attached to these points are exchanged creating new GP-trees.

In the Crossover hill climbing algorithm first described by T. Jones [12,13] only one GP-tree is created from the current solution and a random mate. This offspring is evaluated and replaces the current solution if the fitness is not worse.

We consider the following simple crossover: the *Concatenation Crossover GP* working as follows (see also Algorithm 2). For a fixed population of GP-trees, each GP-tree is chosen to be the parent once. For each parent we choose a mate uniformly at random from the population and create one offspring by *joining* the two trees using a new join-node. Before evaluating the offspring, we employ a local search in the form of the (1+1) GP with bloat control. This local search is performed for a fixed amount of iterations before we discard the GP-tree with worse fitness. The fixed amount depends on the size of the tree and ensures the absence of redundant leaves with high probability (see Lemma 5.2). We note that the amount of redundant leaves depends on the function to be optimized. The functions we studied are variants of MAJORITY, for other functions the amount of iterations ensuring the absence of redundant leaves might be different.

The initial population is generated by creating λ random trees of size s_{init} and employing the local search on each of them. We then proceed in rounds of crossover as described above. We note that we assume all crossover operations to be performed in parallel. Hence, the new population is based entirely on the old population and not partially on previously generated individuals of the new generation.

Algorithm 2: Concatenation Crossover-GP.

```
1  Let LS($t$) denote local search by the (1+1) GP with bloat control on tree $t$ for $90s\log s$ steps, where $s$ is the number of leaves in $t$;
2  for $i = 1$ to $\lambda$ do
3      Let $t_i$ be a random initial tree of size $s_{\text{init}}$;
4      $t_i \leftarrow$ LS($t_i$);
5  while optimum not reached do
6      for $i = 1$ to $\lambda$ do
7          Choose $m \in \{1, \dots, \lambda\} \setminus \{i\}$;
8          $t'_i \leftarrow$ join($t_i, t_m$);
9          $t''_i \leftarrow$ LS($t'_i$);
10         if $f(t''_i) \geq f(t_i)$ then $t_i \leftarrow t''_i$
```

2.2. Terminology

In this section we collect standard theorems on stochastic processes that we will use in the proofs. We start with the Chernoff bound.

Theorem 2.1 *(Chernoff bound). [5] Let $b > 0$. Let the random variables $X_1, \dots, X_n$ be independent and take values in $[0, 1]$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. Then for all $0 \leq \delta \leq 1$,*

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2}$$

and

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/3}.$$

We will also use several drift theorems, which give information about the expected hitting time of a random process if we know the expected progress in each step. The first theorem ("multiplicative drift") captures the case when the expected change is proportional to the current value of X_t.

Theorem 2.2 *(Multiplicative drift, tail bound). [2] Let $(X_\tau)_{\tau\geq 0}$ be a sequence of random variables taking values in $\{0\} \cup [1, \infty)$. Assume that there is a $\delta > 0$ such that*

$$\forall \tau \in \mathbb{N}, x \in \mathbb{N}_0 : \mathbb{E}[X_\tau \mid X_{\tau-1} = x] \leq (1-\delta)x.$$

Then $\tau_0 := \min\{\tau \in \mathbb{N}_0 \mid X_\tau = 0\}$ satisfies

$$\mathbb{E}[\tau_0] \leq \frac{1}{\delta}(\ln(X_0) + 1);$$

$$\forall c > 0,\ \Pr\left[\tau_0 > \frac{1}{\delta}(\ln(X_0) + c)\right] < e^{-c}.$$

We also need analogous theorem for constant additive drift and for variable drift.

Theorem 2.3 *(Additive drift). [10] Let $(X_\tau)_{\tau\geq 0}$ be a sequence of non-negative random variables over a finite state space $S \subseteq \mathbb{R}$. Let τ_0 be the random variable that denotes the earliest point in time $\tau \geq 0$ such that $X_\tau = 0$. If there exists $c > 0$ such that*

$$\mathbb{E}[X_\tau - X_{\tau-1} \mid \tau_0 > \tau] \geq c,$$

then

$$\mathbb{E}[\tau_0 \mid X_0] \leq X_0/c.$$

Theorem 2.4 *(Variable drift). [11] Let $(X_\tau)_{\tau\geq 0}$ be a sequence of non-negative random variables over a finite state space $S \subseteq [0, \infty)$. Let $x_{min} := \min\{x \in S \mid x > 0\}$. Let τ_0 be the random variable that denotes the earliest point in time $\tau \geq 0$ such that $X_\tau = 0$. If there is an increasing function $h : \mathbb{R}^+ \to \mathbb{R}^+$ such that for all $x \in \mathcal{S} \setminus \{0\}$ and all $\tau \geq 0$,*

$$\mathbb{E}[X_\tau - X_{\tau-1} \mid X_{\tau-1} = x] \geq h(x), \tag{1}$$

then

$$\mathbb{E}[\tau_0] \leq \frac{x_{min}}{h(x_{min})} + \int_{x_{min}}^{X_0} \frac{1}{h(x)} dx. \tag{2}$$

Finally we need some tail bounds on the hitting time for additive drift. The following theorem combines two cases in which we have a drift away from zero. Firstly, starting from zero we expect to need time n/c to hit n, and the theorem states that it is exponentially unlikely to need twice as much time. Secondly, starting from n even an exponential number of steps does not suffice to reach zero if the drift pushes in the opposite direction.

Theorem 2.5 *(Tail bounds for additive drift). [18,31,32] Let $(X_\tau)_{\tau\geq 0}$ be a sequence of random variables over $\mathbb{R}$, each with finite expectation. With $\tau_{\geq n} := \min\{\tau \geq 0 \mid X_t \geq n\}$ we denote the random variable describing the earliest point at which the random process exceeds n, and likewise with $\tau_{\leq 0}$ we denote the earliest point at which the random process drops below zero. Suppose there are $c, K > 0$ such that, for all t,*

1. $\mathbb{E}[X_{\tau-1} - X_\tau \mid X_0, \ldots, X_\tau] \geq c$, *and*
2. $|X_\tau - X_{\tau+1}| < K$.

Then there is $\rho > 0$ such that the following is true for all $n \geq 1$.

(a)

$$\Pr\left[\tau_{\geq n} \geq 2n/c \mid X_0 \geq 0\right] \leq e^{-nc/(4K^2)}.$$

(b)

$$\Pr\left[\tau_{\leq 0} \leq e^{\rho n} \mid X_0 \geq n\right] \leq e^{-\rho n}.$$

For the analysis, it will be helpful to partition the set of leaves into three classes as follows. The set $C^+(t) \subseteq S^+(t)$ of *positive critical leaves* is the set of leaves u, whose deletion from the tree results in a decreased fitness. Similarly, the set $C^-(t) \subseteq S^-(t)$ of *negative critical leaves* is the set of leaves u, whose deletion from t results in an increased fitness. Finally, the set $R(t) := [n] \setminus (C^+(t) \cup C^-(t))$ of *redundant leaves* is the set of all leaves u, whose deletion from t does not affect the fitness. Similar as before, we denote $c^-(t) = |C^-(t)|$, $c^+(t) = |C^-(t)|$, and $r(t) = |R(t)|$.

Given a time $\tau \geq 0$, we denote by t_τ the GP-tree after τ iterations of the algorithm. Additionally, we use $S(\tau), s(\tau), S_i(\tau), \ldots$ in order to denote $S(t_\tau), s(t_\tau), S_i(t_\tau), \ldots$. Moreover, we apply the standard Landau notation $O(\cdot)$, $o(\cdot)$, $\Omega(\cdot)$, $\omega(\cdot)$, $\Theta(\cdot)$ as detailed in [1].

The following lemma will be useful for knowing what GP-trees can be expected to be generated in the initial population.

Lemma 2.6. *Let $\nu > 0$ be a constant, and let t be a random GP-tree of size $s(t) = \nu n$. Moreover, let $k, \ell \in \mathbb{N}_0$ be constants, and let $N_{k,\ell}$ be the number of variables i with $s_i^+ = k$ and $s_i^- = \ell$. Then with probability $1 - e^{-\Omega(n^{1/3})}$,*

$$N_{k,\ell} = (1 \pm o(1)) \frac{e^{-\nu}(\nu/2)^{k+\ell}}{k! \cdot \ell!} \cdot n = \Theta(n).$$

In particular, for $+c$-Majority, 2/3-Majority and 2/3-SuperMajority, a constant fraction of the variables are expressed in t, and a constant fraction are not expressed in t.

Proof. Fix a variable i. Each leaf has probability $1/n$ to be an i-literal. Therefore, the number of i-literals out of νn follows approximately a Poisson approximation $\mathrm{Po}(\nu)$. More precisely, if $X_1, \ldots, X_n$ are independent Poisson random variables with parameter ν, and $\mathfrak{E} = \mathfrak{E}(x_1, \ldots, x_n) \in \{0, 1\}$ is any property depending on n integer-valued random variables, then [26, Corollary 5.9],

$$\Pr[\mathfrak{E}(s_1, \ldots, s_n)] \leq 2e\sqrt{\nu n} \Pr[\mathfrak{E}(X_1, \ldots, X_n)]. \tag{3}$$

In particular, by definition of the Poisson distribution, $\Pr[X_i = k+\ell] = e^{-\nu}\nu^{k+\ell}/(k+\ell)! =: q$ independently for every i. Therefore, if we set $Y_i := 1$ if $X_i = k+\ell$, and $Y_i := 0$ otherwise, then $Y := \sum_{i=1}^{n} Y_i$ is the number of random variables with value $k+\ell$. We have $\mathbb{E}[Y] = qn$, and by the Chernoff bound 2.1 with $\delta = n^{-1/3}$,

$$\Pr[Y \notin (1\pm\delta)qn] \le 2e^{-qn^{1/3}/3}.$$

Thus, if we call L the set of $i \in [n]$ such that there are exactly $k+\ell$ i-literals, then by (3)

$$\Pr[|L| \notin (1\pm\delta)qn] \le 2e\sqrt{\nu n}e^{-qn^{1/3}/3} = e^{-\Omega(n^{1/3})}.$$

Therefore, we may assume that $|L| \in (1\pm\delta)q_{k,\ell}n$. Let $L' \subseteq L$ be the set of variables i with $s_i^+ = k$ and $s_i^- = \ell$. Then each $i \in L$ has probability $q' := \binom{k+\ell}{k}/2^{k+\ell} = \Theta(1)$ to be in L', independently for each i. Hence, with $\mu := \mathbb{E}\left[|L'| \,\middle|\, |L| \in (1\pm\delta)qn\right] \ge (1-\delta)q'qn$, a similar application of the Chernoff bound as before shows

$$\Pr\left[|L'| \notin (1\pm\delta)^2 q'qn \,\middle|\, |L| \in (1\pm\delta)qn\right] \le e^{-\delta^2(1-\delta)q'qn/3}.$$

Since $q'q = e^{-\nu}(\nu/2)^{k+\ell}/(k!\ell!)$, the lemma follows.

The remark on expressed variables follows by setting $k = c$ and $\ell = 0$ for $+c$-Majority and $k = 1$ and $\ell = 0$ for the other two functions. The remark on unexpressed variables follows by setting $k = 0$ and $\ell = 0$. □

3. Bloat control

In this section we study how local search with bloat control performs on the given fitness functions. Theorem 3.1 shows that for small initial trees $+c$-Majority cannot be efficiently optimized, while Theorem 3.3 shows that this is possible for 2/3-Majority. Finally, Theorem 3.6 considers 2/3-SuperMajority.

Theorem 3.1. *Consider the (1+1) GP on $+c$-Majority with bloat control on the initial tree with size $s_{\text{init}} < n$. If $c > 1$, with probability equal to 1, the algorithm will never reach the optimum.*

Proof. We assume optimistically that there are no negative leaves in the initial tree, since they slow down the optimization. Let, for all time steps τ, X_τ be the set of variables $i \in [n]$ with no x_i in the tree after step τ. In order to reach the optimum the (1+1) GP has to reach a GP-tree after τ steps with $|X_\tau| = 0$. Since $s_{\text{init}} < n$, there is at least one $j \in [n]$ without x_j in the initial tree, hence $|X_0| \ge 1$. We are going to show that $|X_\tau| \ge 1$ for all τ.

Let $i_0 \in X_0$. Since the bloat control will reject insertions of redundant leaves (in particular of x_{i_0}), the only way to express i_0 is to substitute at least one leaf x_j with x_{i_0}. Furthermore, this substitution cannot reduce the fitness and hence x_j has to be a redundant leaf. If no redundant leaf exists i_0 cannot become expressed. Otherwise, $|X_1| \ge 1$ because either there is only one x_j and by the substitution j is in X_1 or there are more than one x_j and (by counting) $|X_0| \ge 2$. We observe that deletions and insertions can only reduce the number of redundant leaves. Hence, iterating the argument with $i_1 \in X_1$ gives the desired result. □

Theorem 3.2. *Consider the (1+1) GP on 2/3-Majority with bloat control on the initial tree with size $s_{\text{init}} < n$. The expected time until the algorithm computes the optimum is in $\Omega(n \log n)$.*

Proof. By construction, a global optimum has at least one leaf x_i for each variable $i \in [n]$. The set of variables $[n]$ decomposes into four sets:

A: $i \in [n]$ without any leaf x_i or $\overline{x}_i$ in s_{init},
B: $i \in [n]$, such that s_{init} contains a leaf $\overline{x}_i$ but no x_i,
C: $i \in [n]$, such that s_{init} contains a leaf $\overline{x}_i$ and a x_i,
D: $i \in [n]$, such that s_{init} contains a leaf x_i but no $\overline{x}_i$.

Note that variables of type D are already expressed, and variables of type D have an immediate chance to be expressed by a single insertion (of x_i). Variables of type B can only be expressed by a single insertion. Finally, variables of type C may be expressed or not, depending on the ration between leaves of type x_i and of type $\overline{x}_i$.

Since $s_{\text{init}} < n$ the initial tree cannot be the optimal one, furthermore there has to exist a $j \in A$. In order to reach a global optimum, the algorithm needs to add a leaf x_i for every $i \in A$. For every $i \in B$ the algorithm needs to insert x_i as well, but due to the bloat control it will reject such a move if it does not increase the fitness. Hence, the algorithm needs to delete every $\overline{x}_i$ prior to inserting x_i, where the deletion of the last $\overline{x}_i$ can alternatively be done by substituting it with x_i. We observe that, for every $i \in A \cup B$ the algorithm needs to insert x_i or substitute a redundant leaf with x_i. Since doing so is essentially equivalent to a *Coupon Collector*, we obtain a run time of $\Omega(n \log k)$, where k is the cardinality of $A \cup B$. It remains to show, that $n \log k \in \Omega(n \log n)$.

We show $|C \cup D| \leq 2n/3$. For this we note that each entry of the initial tree is a positive leaf with probability $1/2$. Therefore, in expectation, the initial tree contains $s_{\text{init}}/2$ positive leaves. Furthermore, the probability that the amount of positive leaves in the initial tree is higher than the expectation falls exponentially fast. We can observe this behavior due to a Chernoff bound. For all $j \in [s_{\text{init}}]$, let the indicator variable X_j be 1 if the leaf at position j in the initial tree is a positive one. Else, the indicator variable is 0. We obtain

$$\Pr\left[\sum_{j=1}^{s_{\text{init}}} X_j \geq \left(1+\frac{1}{3}\right)\frac{s_{\text{init}}}{2}\right] \leq e^{-s_{\text{init}}/36}$$

Hence, with high probability the initial tree will contain less than $2s_{\text{init}}/3$ positive leaves, yielding $|C \cup D| < 2n/3$ and the claim follows. □

Next we state the upper bound for the performance on 2/3-Majority. The proof of Theorem 3.3 is similar to the one of Theorem 4.1 in [3].

Theorem 3.3. *Consider the (1+1) GP on* 2/3-Majority *with bloat control on the initial tree with size s_{init}. The expected time until the algorithm computes the optimum is in $O(n \log n + s_{\text{init}})$.*

Proof. Let t be a GP-tree over n literals and denote the number of expressed literals of t by $v(t)$. For a best-so-far GP-tree of the (1+1) GP we denote the size of the initial GP-tree by s_{init}. Both parameters n and s_{init} are considered to be given. We partition the set of leaves (again) by observing the behavior when deleting the leaf. This introduces the set of redundant leaves $R(t)$, of critical positive leaves $C^+(t)$ and of critical negative leaves $C^-(t)$ with their respective cardinality denoted by using lower case letters. We obtain

$$s(t) = r(t) + c^+(t) + c^-(t).$$

For $c^-(t)$, observe that the i-th variable can only contribute critical negative leaves if the number of literals x_i is m and the number of literals $\overline{x}_i$ is $\lfloor m/2 \rfloor + 1$ for some $m \geq 1$. In this case, all the positive literals are redundant, and there are at least as many positive literals as negative ones. Since this holds for every i, we obtain $c^-(t) \leq r(t)$. For $c^+(t)$, the i-th variable can only contribute positive critical leaves if there is a unique i-literal, which is positive, or if the number of literals $\overline{x}_i$ is m and the number of literals x_i is $2m$ for some $m \geq 1$. In all cases, the i-th variable contributes at most $2m+1$ positive critical leaves if m is the number of redundant i-literals. Hence,

$$c^-(t) \leq r(t) \quad \text{and} \quad c^+(t) \leq 2r(t) + v(t).$$

For a best-so-far GP-tree t let t' be the GP-tree after one additional round of mutation and selection in the (1+1) GP. By bounding the drift with respect to a suitable potential function g, i.e. the expected change $g(t) - g(t')$ denoted by $\Delta(t)$, we are going to obtain the bound for the optimization time due to the Variable Drift Theorem 2.4. For the potential function, we use the sum of two different non-negative terms, $n - v(t)$ and $s(t) - v(t)$. The idea is that whenever the second term is non-negligible then there are many redundant leaves, which can easily be removed. Thus we obtain a large drift in this case. On the other hand, if the first term dominates then the number of unexpressed variables is much larger than the number of redundant leaves, and hence most of these unexpressed variables are unexpressed because there are no corresponding leaves. In this case, they can be expressed by a single insertion. In the following, we formalize this intuition.

We associate with t the potential $g(t)$ given by

$$g(t) = n - v(t) + s(t) - v(t) = n + s(t) - 2v(t).$$

This potential is 0 if and only if t contains no redundant leaves and for each $i \leq n$ there is exactly one x_i. We observe that the drift cannot be negative since the algorithm only does 1 mutation in each iteration and the bloat control will reject insertions of new redundant leaves.

Case 1: assume $r(t) \geq v(t)$. We obtain

$$s(t) = r(t) + c^+(t) + c^-(t) \leq 5r(t).$$

Let $\mathcal{E}_1$ be the event, that the algorithm deletes a redundant leaf. The drift in this case will be 1 and the probability for such a move is $1/3$ for a deletion followed by at least $1/5$ to choose a redundant leaf. We obtain

$$\mathbb{E}[\Delta(t)] \geq \mathbb{E}[\Delta(t) \mid \mathcal{E}_1]\Pr[\mathcal{E}_1] \geq \frac{1}{15}.$$

Case 2: Suppose $r(t) < v(t)$ and $s(t) \leq n/2$. In particular, we have for at least $n/2$ many $i \in [n]$ that there is neither x_i nor $\overline{x}_i$ present in t. Let $\mathcal{E}_2$ be the event that the algorithm inserts such a x_i. The probability to choose such an x_i is at least $1/4$ and the probability that the algorithm chooses an insertion is $1/3$. We obtain

$$\mathbb{E}[\Delta(t)] \geq \mathbb{E}[\Delta(t) \mid \mathcal{E}_2]\Pr[\mathcal{E}_2] \geq \frac{1}{12}.$$

Case 3: assume $r(t) < v(t)$ and $s(t) > n/2$. In particular, the tree can contain at most $5n$ leaves due to

$$s(t) = r(t) + c^{+}(t) + c^{-}(t) \leq r(t) + 2r(t) + v(t) + r(t) \leq 5v(t) \leq 5n. \tag{4}$$

Hence, the probability that an operation chooses a specific leaf v is

$$\frac{1}{5n} \leq \Pr[\text{choose leaf } v] \leq \frac{2}{n}.$$

Let A be the set of i without x_i or $\overline{x}_i$ in t and let B be the set of i with exactly one x_i but no $\overline{x}_i$ in t. Recall that $R(t)$ is the set of redundant leaves of t. For every j in A let $\mathcal{A}_j$ be the event that the algorithm adds x_j somewhere in t. For every j in $R(t)$, let $\mathcal{R}_j(t)$ be the events that the algorithm deletes j. Finally, let $\mathcal{A}'$, $\mathcal{R}'$ be the event, that one of the $\mathcal{A}_j$, respectively $\mathcal{R}_j(t)$, holds. We obtain

$$\mathbb{E}[\Delta(t) \mid \mathcal{A}_j] = 1 \quad \text{and} \quad \mathbb{E}[\Delta(t) \mid \mathcal{R}_j(t)] = 1,$$

as well as

$$\Pr\left[\mathcal{A}_j\right] \geq \frac{1}{6n} \quad \text{and} \quad \Pr\left[\mathcal{R}_j(t)\right] \geq \frac{1}{15n}.$$

We observe

$$|A| + |R(t)| \geq r(t).$$

Furthermore, we noticed that for any literal j, which is not in B or A, there has to exist at least one redundant leaf x_i or $\overline{x}_i$. We obtain $|A| + |B| + |R(t)| \geq n$ and thus

$$|A| + |R(t)| \geq n - v(t).$$

Additionally, by (4),

$$s(t) - v(t) \leq 4r(t) \leq 4(|A| + |R(t)|),$$

which in conjunction with the above inequality yields

$$5(|A| + |R(t)|) \geq n - v(t) + s(t) - v(t) = g(t).$$

We obtain the expected drift

$$\begin{aligned}
\mathbb{E}[\Delta(t)] &\geq \mathbb{E}[\Delta(t) \mid (\mathcal{A}' \vee \mathcal{R}')]\Pr\left[\mathcal{A}' \vee \mathcal{R}'\right] \\
&= \sum_{j \in A} \mathbb{E}[\Delta(t) \mid \mathcal{A}_j]\Pr\left[\mathcal{A}_j\right] + \sum_{j \in R(t)} \mathbb{E}[\Delta(t) \mid \mathcal{R}_j(t)]\Pr\left[\mathcal{R}_j(t)\right] \\
&\geq |A|\frac{1}{6n} + |R(t)|\frac{1}{15n} \geq \frac{1}{15n}(|A| + |R(t)|) \geq \frac{g(t)}{75n}.
\end{aligned}$$

We distinguish two cases. For $g(t) \leq 5n$, we obtain a multiplicative drift of at least $g(t)/(75n)$, while for $g(t) > 5n$ the drift is at least $1/15$. We now apply the Variable Drift Theorem 2.4 with $h(x) = \min\{1/15, x/(75n)\}$, $X_0 = s_{\text{init}} + n$ and $x_{\min} = 1$, which yields the desired bound on the first time τ_0 such that $g(t_{\tau_0}) = 0$.

$$\begin{aligned}
\mathbb{E}[\tau_0] &\leq \frac{1}{h(1)} + \int_1^{s_{\text{init}}+n} \frac{1}{h(x)}\, dx \\
&= 75n + 75n \int_1^{5n} \frac{1}{x}\, dx + 15 \int_{5n+1}^{s_{\text{init}}+n} 1\, dx \\
&= 75n(1 + \log(5n)) + 15(s_{\text{init}} - 4n - 1) \\
&\leq 75n\log(5n) + 15s_{\text{init}} + 15n.
\end{aligned}$$

This establishes the theorem. □

Corollary 3.4. *Consider the (1+1) GP on* 2/3-Majority *with bloat control on the initial tree with size* $s_{\text{init}} < n$. *The expected time until the algorithm computes the optimum is in* $\mathrm{O}(n \log n)$.

We turn to 2/3-SuperMajority with Theorem 3.6. The proof is based on the following lemma showing that redundant leaves will be removed with sufficient probability. Hence, insertions of positive literals can increase fitness. We remark that the lemma uses non-asymptotic bounds that can be evaluated for any value of n and τ.

Lemma 3.5. *Consider the (1+1) GP on* 2/3-SuperMajority *with bloat control on the initial tree with size* $s_{\text{init}} < n$. *With probability at least* $1 - (\tau/(n\log^2 n))^{-1/(1+4/\sqrt{\log n})}$ *the algorithm will delete any given negative leaf of the initial tree within* $\tau \geq n \log^2 n$ *rounds. For a positive redundant leaf, with the same probability it will either be deleted or turned into a positive critical leaf.*

Proof. Consider any set of b iterations. The expected number of (attempted) insert iterations is $b/3$. Thus, we can use the Chernoff bounds (Theorem 2.1) to bound the size of the tree assuming pessimistically that all insertions are accepted and no deletions are accepted. Thus, for $x \leq b/3$, the growth of the tree over these b iterations is at most $b/3 + x$ with probability at most $\exp\left(-x^2/b\right)$.

We partition the iterations of the algorithm into consecutive blocks of length $b = \log(n)^2$. Let $x = 2\log(n)^{1.5}$. Let $\mathcal{A}_i$ be the event that the size grew, during the ith block, by at most $b/3 + x$. Then for all i, $\Pr\left[\overline{\mathcal{A}_i}\right] \leq \exp\left(-x^2/b\right) = n^{-4}$.

We will henceforth condition on the event $\bigcap_{i=1}^{n^3} \mathcal{A}_i$, which has a probability of at least $1 - 1/n$. Thus, after $i < n^3$ blocks, the tree grew by at most $i(b/3 + x)$. Assume that the designated leaf is either negative or that it does not turn into a non-redundant leaf. By $\mathcal{B}_j$ we denote the event that the leaf is *not* deleted in iteration j. We have, for each iteration j within block i, $\Pr\left[\mathcal{B}_j\right] \leq 1 - 1/(3(s_{\text{init}} + i(b/3 + x)))$. Hence, the probability that the designated leaf is *not* deleted in block i is

$$\Pr\left[\bigcap_{j=bi}^{b(i+1)-1} \mathcal{B}_j\right] \leq \prod_{j=bi}^{b(i+1)-1} (1 - 1/(3(s_{\text{init}} + i(b/3 + x))))$$
$$= (1 - 1/(3(s_{\text{init}} + i(b/3 + x))))^b \leq \exp(-b/(3s_{\text{init}} + ib + 3ix)).$$

We want to compare the denominator with ib. Thus we write $3ix = 3ib/\sqrt{\log n}$. Moreover, for $i \geq n$ we have $3s_{\text{init}} \leq 3n \leq ib/\sqrt{\log n}$. Hence, the probability of *not* deleting the designated leaf with the first $\tau/b < n^3/b$ blocks is at most

$$\prod_{i=1}^{\tau/b} \exp(-b/(3s_{\text{init}} + ib + 3ix)) \leq \prod_{i=n}^{\tau/b} \exp(-b/(ib + 4ib/\sqrt{\log n}))$$
$$= \exp\left(-(1/(1 + 4/\sqrt{\log n})) \sum_{i=n}^{\tau/b} 1/i\right) \leq \exp\left(-\ln(\tau/(bn))/(1 + 4/\sqrt{\log n})\right)$$
$$= (\tau/(bn))^{-1/(1+4/\sqrt{\log n})}. \quad \square$$

Theorem 3.6. *Consider the (1+1) GP on* 2/3-SuperMajority *with bloat control on an initial tree with size* $s_{\text{init}} < n$, *and let* $\varepsilon > 0$. *Then, the algorithm will express every literal after* $n^{2+\varepsilon}$ *iterations with probability* $1 - \mathrm{o}(1)$.

Proof. Consider the set A of leaves which are redundant or negative in the initial tree. Due to bloat control, the number of such leaves can only decrease over time. We call a leaf $i \in A$ *bad*, if it is not deleted or turned into a positive critical leaf in the first half of the iterations. By Lemma 3.5, each fixed negative leaf in A has probability at most $1 - (n/(2\log^2 n))^{-(1+\varepsilon)/(1+4\sqrt{\log n})}$ of being deleted within the first $\tau = n^{2+\varepsilon}/2$ iterations. If n is sufficiently large, then $(1+\varepsilon)/(1+4\sqrt{\log n}) \geq 1 + \varepsilon/2$. Therefore, the probability that the leaf is bad is at most $(n/(2\log^2 n))^{-1-\varepsilon/2} = \mathrm{o}(1/n)$. By a union bound, the probability that there is *any* bad leaf is $\mathrm{o}(1)$. In particular, with probability $1 - \mathrm{o}(1)$, after the first half of the iterations there are no negative leaves left.

Without negative literals, in the second half of the $n^{2+\varepsilon}$ iteration, an unexpressed variable has probability $1/(6n)$ to become expressed in each step. Therefore, the probability that it is not expressed after the second half is at most $(1 - 1/(6n))^{n^{2+\varepsilon}/2} = \mathrm{o}(1/n)$. Thus the expected number of unexpressed variables is $\mathrm{o}(1)$. By Markov's inequality, with probability $1 - \mathrm{o}(1)$ all variables are expressed after the second half of the algorithm, as claimed. $\square$

4. No bloat control

In this section we study the fitness function 2/3-SuperMajority, which facilitates bloat of the GP-tree.

Theorem 4.1. *For any constant $\nu > 0$, consider the (1+1) GP without bloat control on $2/3$-SuperMajority on the initial tree with size $s_{\text{init}} = \nu n$. There is $\varepsilon = \varepsilon(\nu) > 0$ such that, with probability $1 - o(1)$, an ε-fraction of the indices will never be expressed. In particular, the algorithm will never reach a fitness larger than $(2 - 2\varepsilon)n$.*

We commence with some preparatory lemmas before proving the theorem. First, we analyze how the size of the GP-tree evolves over time. We recall that $s(\tau)$ is the number of leaves of the GP-tree at time τ.

Lemma 4.2. *There is a constant $0 < \eta \le 1$ such that, with probability $1 - o(1)$, for all $\tau \ge 0$ we have $s(\tau) \ge \eta\tau$.*

Proof. Let $v(\tau)$ be the number of expressed literals at time τ. By Lemma 2.6, with high probability there are at least $\eta' n$ expressed variables in the initial tree, for some constant $\eta' > 0$. So $v(0) \ge \eta' n$.

Now we examine how the number $c^+(\tau)$ of positive critical leaves evolves over time. We claim that with high probability, for all $\tau \ge 0$,

$$c^+(\tau) \ge \eta'\tau/12. \tag{5}$$

Note that a mutation that decreases the number of expressed literals also decreases the fitness and is rejected. Thus $v(\tau)$ is increasing in τ, and hence $c^+(\tau) \ge v(\tau) \ge v(0) \ge \eta' n$ for all $\tau \ge 0$. This already implies (5) for $\tau \le 12n$. Similarly, an offspring in which a critical positive leaf is deleted is never accepted, and an offspring in which a critical positive leaf is substituted can only be accepted if the leaf is substituted by another positive critical leaf. Therefore, $c^+(\tau)$ is also increasing in τ. Moreover, in each step of the algorithm, with probability $v(\tau)/(6n) \ge \eta'/6$ a new positive literal is created that is already expressed. In this case, the number of positive critical literals increases by one. Hence, for all $\tau \ge 0$,

$$\mathbb{E}[c^+(\tau+1) - c^+(\tau)] \ge \eta'/6. \tag{6}$$

For any fixed $\tau \ge 12n$, by the tail bounds on the Additive Drift Theorem 2.5, $\Pr[c^+(\tau) \le \eta'\tau/12] \le e^{-\rho\tau}$, where $\rho > 0$ is the constant from Theorem 2.5. Therefore, by a union bound,

$$\Pr[\exists \tau \ge 12n \mid c^+(\tau) \le \eta'\tau/12] \le \sum_{\tau=12n}^{\infty} e^{-\rho\tau} = o(1).$$

This proves (5) for all $\tau \ge 0$. The lemma now follows simply from $s(\tau) \ge c^+(\tau)$ with $\eta := \eta'/12$. □

In order to continue we need some more terminology. For an index $i \in [n]$, we recall that $s_i^+(\tau)$ and $s_i^-(\tau)$ denote the number of x_i- and $\overline{x}_i$-literals at time τ, respectively, and $s_i(\tau) := s_i^+(\tau) + s_i^-(\tau)$. We call index i *touched* in round τ, if a literal x_i or $\overline{x}_i$ is deleted, inserted or substituted, or if a literal is substituted by x_i or $\overline{x}_i$. We call the touch *increasing* if it is either an insertion or if a literal is substituted by x_i or $\overline{x}_i$. We call the touch *decreasing* if it is a deletion or substitution of a x_i or $\overline{x}_i$ literal. We note that in exceptional cases a substitution may be both increasing and decreasing. Let $\rho_i(\tau)$ be the number of increasing touches of i up to time τ. We call a decreasing step *critical* if it happens at time τ with $s_i(\tau) \le \eta\tau/(4n)$, and we call $\gamma_i(\tau)$ the number of critical steps up to time τ. Finally, we call a round *accepting* if the offspring is accepted in this round.

The approach for the remainder of the proof is as follows. First, we will show that in the regime, where critical steps may happen (i.e., $s_i(\tau) \le \eta\tau/(4n)$), it is more likely to observe increasing than decreasing steps. The reason is that a step is only critical if there are relatively few i-literals, in which case it is unlikely to delete or substitute one of them, whereas the probability to insert an i-literal is not affected. It will follow that $s_i(\tau)$ grows with τ, since otherwise we would need many critical steps. Finally, if $s_i(\tau)$ keeps growing it becomes increasingly unlikely to obtain a $2/3$ majority. In order to state the first points more precisely we fix a $j_0 \in \mathbb{N}$ and call an index i *bad* (or more precisely, j_0-bad) if the following conditions hold: for all $\tau \ge j_0 n$ and $\tau_0 := j_0 n$

(A) $s_i^+(\tau_0) \le s_i^-(\tau_0) \le j_0$ (B) $\tau/(2n) \le \rho_i(\tau) \le 2\tau/n$
(C) $\gamma_i(\tau) \le 2\tau/n$ (D) $s_i(\tau) \ge \eta\tau/(8n)$.

In particular, in (A) x_i is not expressed at time τ_0.

Lemma 4.3. *For every fixed $i_0 > 0$, with probability $1 - o(1)$ there are $\Omega(n)$ bad indices.*

Proof. We will show that a given index i has probability $\Omega(1)$ to be bad. It is more technical to show concentration, and we only give a sketch of the argument at the end of the proof. So fix an index i. We will first show individually[1] that (B),

[1] with some slight complications for (D).

(C), (D) all hold with rather large probability, say, with probability 0.9. Then by a union bound, the probability that one of them does *not* hold is at most $0.3 < 1$. Finally we show that (A) holds with probability $\Omega(1)$, and that (B), (C), (D) still hold with sufficiently large probability if we condition on (A).

We start with (B). Note that the probability for a round to be increasing is always $2/(3n)$, independent of the current GP-tree. In particular, it is independent of whether x_i is expressed or not. Therefore, the expected number of increasing rounds up to time τ is $2\tau/(3n) = 6\tau/(12n)$. Assume that for some τ_j of the form $\tau_j = (6/5)^j j_0 n$ the inequality $\rho_i(\tau_j) \geq 5\tau_j/(12n)$ holds, which is stronger than the first inequality in (B). Then for all $\tau' \in [\tau_j, \tau_{j+1}]$ it will follow that $\rho_i(\tau) \geq \rho_i(\tau_j) \geq 5\tau_j/(12n) \geq \tau/(2n)$, as in (B). Therefore, it suffices to show that $\rho_i(\tau_j) \geq 5\tau_j/(12n)$ holds for all $j \in \mathbb{N}$ to conclude the first inequality in (B). However, by the Chernoff bound the probability that this is violated for any large j is

$$\Pr[\exists j \geq j_0 \mid \rho_i(\tau_j) \leq 5\tau_j/(12n)] \leq \sum_{j \geq j_0} e^{-\tau_j/50} = e^{-\Omega(i_0)},$$

which is small if j_0 is sufficiently large. An analogous argument shows the second inequality $\rho_i(\tau) \leq 2\tau/n$ of (B). Thus, every index i has large probability (at least 0.9) to satisfy (B).

For (C), essentially the same argument applies again. By definition a critical round can only occur if $s_i(\tau) \leq \eta\tau/(4n)$. By Lemma 4.2 this implies $s_i(\tau) \leq s(\tau)/(4n)$, so the probability to choose a deletion or substitution that hits an i-literal is at most $2/3 \cdot 1/(4n) = 1/(6n)$. The rest follows as for (B), with room to spare.

For (D), assume that for some $\tau \geq j_0 n$ we have $s_i(\tau) \leq \eta\tau/(4n)$. Then as for (C), the probability that a step is decreasing is at most $1/(6n)$, while the probability of an increasing step is $2/(3n)$. Therefore, if we consider the random variable $X(\tau) := s_i(\tau) - \eta\tau/(4n)$ then $X(\tau)$ has a *positive* drift whenever $X(\tau) \leq 0$,

$$\mathbb{E}[X(\tau+1) - X(\tau) \mid X(\tau) \leq 0] \geq \frac{1}{2n} - \frac{\eta}{4n} \geq \frac{1}{4n}. \tag{7}$$

We claim that this renders it unlikely that $s_i(\tau) < \eta\tau/(8n)$ for some $\tau \geq j_0 n$. Indeed, assume that such a $\tau = \tau_0$ exists, and let $\tau_0' := \max\{\tau \in [j_0 n, \tau_0] \mid X(\tau) \geq 0\}$ be the last point in time at which $X(\tau)$ was non-negative. (Assume for the moment that $X(j_0 n) \geq 0$ so that such a time exists.) For technical reasons that will become clear later, let τ_0'' be the time of the first decreasing step *after* τ_0'. Then from τ_0'' to τ_0, $X(\tau)$ performs a random walk with positive drift, which declines from $X(\tau_0'') \geq -1$ to $X(\tau_0) < -\eta\tau_0/(8n) < -\eta\tau_0''/(8n)$ without hitting $X(\tau) \geq 0$ in the meantime. However, for any fixed τ_0'', this happens with probability at most $e^{-\Omega(\tau_0''/n)}$ by Theorem 2.5. Moreover, assume for a moment that (C) holds. Then since τ_0'' is a critical step, there are only a limited number of candidates for τ_0'', because the j-th critical step does not happen before $\tau_j = jn/2$. Hence, the probability to have a random walk that declines from $X(\tau_0'') \geq -1$ to $X(\tau_0) < -\eta\tau_0''/(8n)$ for some $\tau_0'' \geq j_0 n$ is at most $\sum_{j \in \mathbb{N}, \tau_j \geq j_0 n} e^{-\Omega(\tau_j/n)} = e^{-\Omega(j_0)}$. Therefore, the only possibilities that (D) fails with some $\tau_0'' \geq j_0 n$ is that either (C) fails (which is unlikely), or that there is a strongly declining random walk (which is also unlikely). This shows that (B), (C), (D) all happen with probability at least 0.9 if we assume that $X(j_0 n) \geq 0$.

It remains to argue that it is sufficiently likely that both $X(j_0 n) \geq 0$ and (A) holds. Consider the event that there are exactly j_0 increasing and no decreasing steps until time $j_0 n$, and that all the increasing steps until time $j_0 n$ introduce negative literals. This event has probability $\Omega(1)$, and it implies $X(j_0 n) \geq 0$ and (A). Moreover, each of (B), (C), (D) still holds with probability at least 0.9 if we condition on this event. Therefore, (A), (B), (C), (D) all hold simultaneously with probability $\Omega(1)$.

Altogether, we have shown that a literal i has probability $\Omega(1)$ to be bad. Therefore, the expected number of bad literals is $\Omega(n)$. It remains to show concentration, for which we only give a sketch. We would like to use a concentration bound like the Chernoff bound, but unfortunately for two indices i and i' the events "i is bad" and "i' is bad" are not independent. For example, if we add a x_i literal at time τ, then we cannot add a $x_{i'}$ literal in the same iteration. However, we can couple the process to an uncovering process with independent steps as follows. Consider a random set $A \subseteq [n]$ of size εn, for some small constant $\varepsilon > 0$. Then by the Chernoff bound, the literals corresponding to A will constitute at most a 2ε fraction of the leaves in the GP-tree, and they will only affect a 2ε fraction of all the iterations. Here we use implicitly that there are $\Omega(n)$ expressed literals from the beginning, and thus there is no single literal which constitutes a large fraction of the leaves. Then for the indices in A, we reveal one by one whether they are bad or not. The crucial advantage is that even after uncovering for the first indices in A whether they are bad, the remaining indices still have probability $\Omega(1)$ to be bad. The reason is that our proof that the probability is $\Omega(1)$ is still valid if we have information about a 2ε fraction of the rounds. Thus we may couple the process of uncovering to a process where we flip independent coins for each $i \in A$, and the Chernoff bound tells us that the number of bad indices in A is concentrated. We omit the details. □

Lemma 4.4. *Every bad index has probability $\Omega(1)$ that it is never expressed, independent of the other bad indices.*

We note that Lemmas 4.3 and 4.4 imply Theorem 4.1 by a straightforward application of the Chernoff bound.

Proof of Lemma 4.4. Let i be a bad index. For $j \geq 0$, let τ_j be the j-th accepting round after $j_0 n$ in which i is touched. (We note that the i-literals have no effect on the fitness while i is not expressed. However, an offspring may be rejected in substitutions.) We will study how $\delta(j) := s_i^+(\tau_j) - s_i^-(\tau_j)$ evolves over time.

We first show that if $\delta(j)$ is not too large ($\delta(j) < \eta j/144$) then i is not expressed. Initially (at time $j_0 n$), by (A) there are at most $2j_0$ i-literals. Since the number of i-literals is non-negative, the number of decreasing accepting rounds exceeds the number of increasing accepting rounds by at most $2j_0$. Additionally, by (B) the number of increasing accepting rounds before time $\tau \geq j_0 n$ is at most $2t/n$. Therefore, the total number of accepting rounds that touch i before time $\tau \geq j_0 n$ is at most $4\tau/n + 2j_0 \leq 6t/n$. In particular, this implies $\tau_j \geq jn/6$. Due to (D) we have $s_i(\tau_j) \geq \eta\tau_j/(8n) \geq \eta j/48$. Thus, if $s_i^+(\tau_j) \geq \frac{2}{3}s_i(\tau_j)$ this implies $\delta(j) \geq \frac{1}{3}s_i(\tau) \geq \eta j/144$. Conversely, if $\delta(j) < \eta j/144$ then i is not expressed.

We proceed by studying how $\delta(j)$ evolves over time. In order to avoid border cases we will show that, with probability $\Omega(1)$, we have $\delta(j) \leq \eta j/144 - 1$ for all $j \in \mathbb{N}$. Moreover, we will treat a substitution that changes $\delta(j)$ by 2 as two consecutive operations. We note that in this regime i cannot become expressed by a single step. Thus, the selection operator does not discriminate between x_i and $\overline{x}_i$. Regardless of $\delta(j)$, increasing operations have the same probability to introduce x_i and $\overline{x}_i$. Regarding decreasing operations, if $\delta > 0$ then it is more likely to select a x_i-literal than a $\overline{x}_i$-literal (because there are more x_i-literals than $\overline{x}_i$-literals). Hence, it is more likely to decrease δ than to increase it. Likewise, for $\delta < 0$ it is more likely to increase δ than to decrease it. Therefore, $\delta(j)$ performs a random walk with $\delta(j+1) = \delta_j \pm 1$ and

$$\Pr[\delta(j+1) = \delta(j)+1] \geq 1/2, \text{ if } \delta < 0;$$
$$\Pr[\delta(j+1) = \delta(j)+1] \leq 1/2, \text{ if } \delta > 0.$$

Therefore, for any $k \geq 0$ the probability that $|\delta(j)| > k$ is at most the probability that an unbiased random walk takes a value $> k$ after j steps. This latter probability is $2\Pr[\mathrm{Bin}(j, 1/2) \geq j/2 + k/2]$, where Bin is the binomial distribution. In particular, for $k = \eta j/144 - 1$

$$\Pr[|\delta(j)| \geq k] \leq 2\Pr[\mathrm{Bin}(j, 1/2) \geq j/2 + k/2] = e^{-\Omega(k)} = e^{-\Omega(j)},$$

where the last step follows from the Chernoff bound. Due to a union bound over all $j \geq j_1$ the probability that there is $j \geq j_1$ with $|\delta(j)| \geq \eta j/144 - 1$ is $e^{-\Omega(j_1)}$. Therefore, it becomes more and more unlikely that the literals ever becomes expressed. It remains to choose (somewhat arbitrarily) a sufficiently large constant j_1 and to observe that, with probability $\Omega(1)$, we have $\delta < 0$ in the first j_1 rounds. This concludes the proof. □

5. Crossover

In the following we will study the performance of the Concatenation Crossover GP (Algorithm 2) on $+c$-MAJORITY and 2/3-MAJORITY with bloat control. As observed in Theorem 3.1 the (1+1) GP with bloat control may never reach the optimum when optimizing an initial tree of size $s_{\text{init}} < n$. We will deduce that crossover solves this issue and the algorithm reaches the optimum fast. We commence this section by stating the exact formulation of said result in Theorem 5.1 followed by an outline of its proof. Finally, we show the corresponding result for 2/3-SUPERMAJORITY in Theorem 5.5.

Theorem 5.1. *Consider the Concatenation Crossover GP on $+c$-MAJORITY or 2/3-MAJORITY with bloat control on the initial tree with size $2 \leq n/2 \leq s_{\text{init}} \leq b\,n$ (for a constant $b > 1/2$). Then there is a constant $c_\lambda > 0$ such that for all $n^2 \geq \lambda \geq c_\lambda \log n$, with probability in $(1 - O(n^{-1}))$, the algorithm reaches the optimum after at most $O(n\log^3(n))$ steps.*

Before we will prove the theorem we are going to state two auxiliary lemmas. First, Lemma 5.2 states the absence of redundant leaves in a GP-tree t after the local search with a probability of $1 - n^{-5}$. This will be applied after every local search. We observe for two GP-trees t_1 and t_2 *without redundant leaves*: if t' is the tree resulting from joining t_1 and t_2, then a variable $i \in [n]$ is expressed in t' if and only if it is expressed in t_1 or t_2.

Second, Lemma 5.3 states that, with a probability of $1 - n^{-5}$, each variable $i \in [n]$ is expressed in at least one of $\lambda/2$ trees before the first crossover. Combining both lemmas, for a fixed GP-tree t it will suffice to observe the time until t has been joined with at least $\lambda/2$ different trees.

Lemma 5.2. *Consider the (1+1) GP with bloat control on either $+c$-MAJORITY or 2/3-MAJORITY. For an initial tree with size $2 \leq n/2 \leq s_{\text{init}} \leq bn$ (for constant $b > 0$) after $90s_{\text{init}}\log(s_{\text{init}})$ iterations, with probability at least $1 - n^{-5}$, the current solution will have no redundant leaves.*

Proof. Given a GP-tree t we define $r(t)$ to be the number of redundant leaves in t. Similarly, we define $s(t)$ to be the number of leaves in t. By applying the Multiplicative Drift Theorem 2.2 we are going to derive a bound on the expected time τ until the $(1+1)$ *GP* has for the first time sampled a solution t with $r(t) = 0$. Additionally, the theorem will yield that τ will not be significantly larger than its expectation with high probability.

Given the best-so-far solution t let t' be the next GP-tree after one round of mutation and selection in the (1+1) GP. In order to apply Theorem 2.2 we need to derive an upper bound on $\mathbb{E}[r(t') \mid r(t)]$. First, we observe that due to the bloat control t' cannot have more redundant leaves than t. Additionally, the difference will be 1 if the algorithm chose a redundant leaf and deleted it; we denote this event as $\mathcal{A}$. Let $\overline{\mathcal{A}}$ be the complementary event of $\mathcal{A}$. Due to our observations and the law of total expectation we already have

$$\begin{aligned}\mathbb{E}[r(t') \mid r(t)] &= \mathbb{E}[r(t') \mid r(t), \mathcal{A}]\Pr[\mathcal{A}] + \mathbb{E}[r(t') \mid r(t), \overline{\mathcal{A}}]\Pr[\overline{\mathcal{A}}] \\ &\le \left(1 - \frac{1}{3s(t)}\right) r(t)\end{aligned}$$

since the probability to choose a redundant leaf is $r(t)/s(t)$ followed by the probability of $1/3$ for a deletion. In order to further bound the drift, we need an upper bound on the size of the GP-tree during the optimization. Due to the bloat control the worst-case for a growth of the size is an initial tree, where the insertion of x_i will yield the expression of the variable i for all $i \in [n]$. Therefore, $s(t) \le n + s_{\text{init}} \le 3s_{\text{init}}$ and, since this bound is valid for both functions $+c$-Majority and $2/3$-Majority, we obtain

$$\mathbb{E}[r(t') \mid r(t)] \le \left(1 - \frac{1}{9s_{\text{init}}}\right) r(t).$$

Applying the Multiplicative Drift Theorem 2.2 on $r(t) \in \{0\} \cup [1, s_{\text{init}}]$ with initial tree t_0 yields that the time τ to remove all redundant leaves satisfies

$$\mathbb{E}[\tau \mid r(t_0)] \le 9s_{\text{init}}(\log(s_{\text{init}}) + 1).$$

Moreover, we also obtain for every $x > 0$

$$\Pr[\tau > 9s_{\text{init}}(\log(s_{\text{init}}) + x)] \le e^{-x}.$$

Therefore, for $x = 9\log(s_{\text{init}})$ we obtain that the solution at iteration $\tau = 90s_{\text{init}}\log(s_{\text{init}})$ will have no redundant leaves with probability at least $1 - s_{\text{init}}^{-9} \ge 1 - (n/2)^{-9} \ge 1 - n^{-5}$. □

Lemma 5.3. *Consider the Concatenation Crossover GP on $+c$-Majority or $2/3$-Majority with bloat control on initial trees with size $2 \le n/2 \le s_{\text{init}} \le bn$ (for constant $b > 0$). Then there is a constant $c_\lambda > 0$ such that for all $\lambda \ge c_\lambda \log n$, with probability at least $1 - n^{-5}$, each variable will be expressed in at least one of $\lambda/2$ trees before the first crossover.*

Proof. Given a GP-tree t, for each variable $i \in [n]$ let $\mathcal{A}_i$ be the event that i is expressed in t after initialization. By Lemma 2.6 we have $\Pr[\mathcal{A}_i] \ge c^*$ for a constant $c^* > 0$. In order to utilize this bound after the Local Search too we observe that the probability $\Pr[\mathcal{A}_i]$ is independent of the choice of i. Since the number of expressed variables cannot decrease during Local Search, the same bound of c^* also applies after the Local Search.

For a variable $i \in [n]$ let $\mathcal{E}_i$ be the event, that i is expressed in at least one of $\lambda/2$ trees after the Local Search. Hence, the complementary event $\overline{\mathcal{E}}_i$ implies that i is not expressed in each of $\lambda/2$ trees. Due to the independence of the trees from each other,

$$\begin{aligned}\Pr\left[\overline{\mathcal{E}}_i\right] &\le (1 - \Pr[\mathcal{A}_i])^{\lambda/2} \le (1 - c^*)^{\lambda/2} \\ &\le e^{-c_\lambda \log(n) c^*/2} = n^{-c_\lambda c^*/2}. \end{aligned} \tag{8}$$

Let $\mathcal{E}$ be the event that each variable $i \in [n]$ is expressed in at least one of $\lambda/2$ trees after the Local Search. Therefore, the complementary event $\overline{\mathcal{E}}$ implies that any variable i is not expressed in each of $\lambda/2$ trees. $\overline{\mathcal{E}}$ decomposes into the events $\overline{\mathcal{E}}_i$ and the union bound together with (8) yields

$$\Pr\left[\overline{\mathcal{E}}\right] = \Pr\left[\bigcup_{i=1}^{n} \overline{\mathcal{E}}_i\right] \le \sum_{i=1}^{n} \Pr\left[\overline{\mathcal{E}}_i\right] \le n\, n^{-c_\lambda c^*/2} = n^{1 - c_\lambda c^*/2}.$$

The only part remaining is to choose c_λ such that $c_\lambda c^*/2 \ge 6$, which implies the desired result $\Pr[\mathcal{E}] \ge 1 - n^{-5}$. □

Using these two lemmas we can now prove Theorem 5.1.

Proof (of Theorem 5.1). As explained in the beginning of this section, we will apply Lemma 5.2 after each local search, yielding the absence of redundant leaves. Second, Lemma 5.3 yields that each variable $i \in [n]$ is expressed in at least one of $\lambda/2$ trees. Finally, we will study the time until a fixed GP-tree t has been joined with at least $\lambda/2$ different trees. Utilizing the Additive Drift Theorem 2.3 with the corresponding Tail Bounds Theorem 2.5 will yield the desired optimization time as well as the probability for it to hold.

We commence by studying the probability for the event $\mathcal{A}$, that no redundant leaf is left in any of the λ trees after local search. Due to Lemma 5.2 we have that for a GP-tree t $\Pr[r(t) = 0 \text{ after LS}] \ge 1 - n^{-5}$. Thus, we obtain

$$\Pr[\mathcal{A}] \geq \left(1-n^{-5}\right)^{\lambda} \tag{9}$$

which still approaches 1 rapidly due to $\lambda = c_\lambda \log(n)$. Let us assume that the event $\mathcal{A}$ holds and let t_r and t_j be two GP-trees after Local Search, which are going to be joined with crossover together yielding the tree t_r'. Due to the absence of redundant leave a variable $i \in [n]$ is expressed in t_r' if and only if it is expressed in t_r or t_j. Hence, for a sufficiently large subset L of the λ trees, where each variable is expressed in at least one of the trees in L, it suffices to study the time until one tree t has been joined with every tree in L. The resulting tree t will have every variable expressed and, after Local Search, no redundant leaf left. Lemma 5.3 already gives us that any subset L of size at least $\lambda/2$ satisfies the desired property with a high probability.

Let $\mathcal{B}$ be the event that no redundant leaf is left in any of the λ trees after Local Search and each variable is expressed in at least one of $\lambda/2$ trees. As explained above we obtain

$$\Pr[\mathcal{B}] \geq \left(1-n^{-5}\right)\left(1-n^{-5}\right)^{\lambda}. \tag{10}$$

Let us assume that the event $\mathcal{B}$ holds. We will now study the expected time until a fixed GP-tree t of the λ trees after the initial Local Search has been joined with at least $\lambda/2$ trees, which is essentially a Coupon Collector process. This will yield the expected number of crossover cycles τ until t will be optimal. Additionally, similar to the proof of Lemma 5.2 we obtain that τ will not be significantly larger than its expectation with high probability. At the end we derive the probability for all the assumed events to hold.

For a GP-tree t let $v(t) \in [0, \lambda/2]$ be the number of different GP-trees joined with t. We define

$$g(t) = \lambda/2 - v(t),$$

which measures the remaining GP-trees until t has been joined with at least $\lambda/2$ different trees. Let t' be the offspring of t by joining t with a random GP-tree t_j with $j \in [\lambda]$. We obtain for the expected difference between parent and offspring

$$\mathbb{E}[g(t) - g(t') \mid g(t)] \geq v(t)/\lambda \geq 1/2,$$

since we always have a probability of at least 1/2 to choose a tree t_j, whom we did not join with so far. The Additive Drift Theorem 2.3 yields for the initial tree t_0 and the time τ until we joined t_0 with at least $\lambda/2$ different trees

$$\mathbb{E}[\tau \mid g(t_0)] \leq \lambda.$$

Therefore, we need an expected amount of λ cycles until one tree t will be optimal. Moreover, due to the constant step-size of at most 1 and the finite expected difference of at least $\varepsilon = 1/2$ we are allowed to apply Theorem 2.5 and obtain for $x \geq \lambda$ that $\Pr[\tau \geq x] \leq e^{-x/32}$. We note that the statement of Theorem 5.1 becomes weaker for larger c_λ. Therefore, for $\lambda \geq c_\lambda \log n$ we may assume $c_\lambda \geq 96$ yielding

$$\Pr[\tau \geq 2\lambda] \leq n^{-6}. \tag{11}$$

What remains is to sum up the runtime for each cycle and derive the probability that t will be optimal.

Each cycle consists of λ crossover operations, which contain a Local Search of fixed time. Due to the absence of redundant leaves we know that every tree t will have size $|t| \leq cn$ during the crossover. Therefore, said Local Search will be of fixed time at most $90cn\log(cn)$ and each crossover cycle consists of at most $\lambda 90cn\log(cn)$ steps. Due to (11) we need 2λ cycles with a high probability. Thus, with the probability to be deduced below, the time τ' until the crossover computes an optimal tree t for the first time is

$$\tau' \leq 180\, cn\, \lambda^2 \log(cn) \in \mathrm{O}(n \log^3(n)).$$

This is asymptotically larger than the time for the initial cycle of Local Searches, which consist of at most $\lambda\, 90\, bn \log(bn) \in \mathrm{O}(n\log^2(n))$ steps.

What remains is to calculate the probabilities for the bound to hold. Let $\mathcal{C}$ be the event that $\mathcal{B}$ holds and the Local Search for the tree t expressing all variables yields a tree without redundant leaves. The latter event holds, if every Local Search of each relevant tree up to and including said point yields a tree without redundant leaves. During the crossover there are $\lambda/2$ relevant trees in each crossover cycle. Applying (9) and (10) we obtain

$$\Pr[\mathcal{C}] \geq \Pr[\mathcal{B}]\left(\Pr[\mathcal{A}]^{\lambda/2}\right)^{2\lambda} \geq \left(1-n^{-5}\right)^{\lambda^2+\lambda+1}.$$

In conjunction with the probability that we need at most 2λ crossover cycles given by (11) this yields the desired probability of

$$\left(1-n^{-6}\right)\Pr[\mathcal{C}] \geq \left(1-n^{-6}\right)\left(1-n^{-5}\right)^{\lambda^2+\lambda+1} \geq \left(1-n^{-5}\right)^{(\lambda+1)^2}.$$

Since $\lambda \leq n^2$ this concludes the proof. □

Finally, we turn to 2/3-SuperMajority. For the proof we use a result from the area of rumor spreading relating to the *pull protocol* [14,25] in order to study the time until every individual of the population has every variable expressed. The idea here is similar to the previous proofs with crossover: expressed variables can be collected with crossover. For this purpose we need to show that after a crossover between an individual which has variable i expressed and an individual that does *not* have i expressed, the offspring will have i expressed. To this end, we need to show that the number of literals x_i in the first parent (which expresses i) is larger than the number of literals $\overline{x}_i$ in both parents together.

Lemma 5.4 *(Pull-protocol [14,25]). Consider the situation where n agents act in rounds in order to gain an information; suppose initially a constant fraction of all agents hold that information. In each round, each uninformed agent selects another agent uniformly at random and is now considered informed if the selected agent is informed. Then, within $\mathrm{O}(\log\log n)$ iterations, all agents are informed with probability at least $1-n^{-c}$, for any given c.*

Theorem 5.5. *Consider the Concatenation Crossover GP with bloat control with initial tree size $s_{\text{init}} = n/2$ on 2/3-SuperMajority. Then, there is a constant $c_\lambda > 0$ such that, for $\lambda = c_\lambda \log n$, each GP-tree in the population has all variables expressed after at most $\mathrm{O}(n^{1+o(1)})$ steps with probability at least $1-\mathrm{O}(n^{-4})$.*

Proof. Fix any given variable i. We will show that i is expressed in all individuals after $\log\log n$ rounds of crossover with probability at least $1-n^{-5}$, so that a union bound gives the desired probability (leaving the bound on the number of steps to be shown). In the following we reason for n large enough.

Using a Chernoff bound we see that, with probability at least $1-n^{-6}$, the maximum number of $\overline{x}_i$ in a GP-tree after initialization is at most $20\log n/\log\log n$. Note that local search may introduce $\overline{x}_i$, but only by substituting from a redundant leaf, since we use bloat control. Since there are at most n redundant leaves, if the GP-tree is of size s, we gain an $\overline{x}_i$ with probability at most $1/(6s)$, while deleting any given $\overline{x}_i$ with probability at least $1/(3s)$. Again employing a Chernoff bound we see that, with probability at least $1-n^{-6}$, the maximum number of $\overline{x}_i$ in a GP-tree after local search is at most $40\log n/\log\log n$. Thus, by induction (and analogous reasoning for future iterations) and using that all crossover steps are based on the old population, after $k \geq 0$ rounds of crossover every individual of the population has at most $2^k \cdot 40\log n/\log\log n$ occurrences of $\overline{x}_i$.

Consider now any individual where i is *expressed* and where at least n/c variables are expressed, for some c; let s be the size of that individual. Note that $s \geq n/c$. Another application of the Chernoff bound gives that, after local search is applied to this individual for $90s\log s$ iterations, we have at least $s\log(s)/n$ many x_i with probability $1-n^{-6}$, while we get for n sufficiently large that the tree grows by at least $2s\log(s)/\log\log n \geq s\log n/\log\log n$ with probability at least $1-n^{-6}$ and by at most $90s\log s$ with certainty. Induction gives that, after k rounds of crossover, with probability at least $1-2kn^{-6}$, the total size of any such GP-tree is at least $n(\log n/\log\log n)^k$ and the number of x_i is at least $(\log n)^{k+1}$.

Thus, whenever we mate two individuals where one has i expressed and the other does not, the offspring has i expressed: the number of x_i in the individual with i expressed is at least $(\log n)^{k+1}$ and thus larger than the number of $\overline{x}_i$ in any mate, which is at most $2^k \cdot 40\log n/\log\log n$.

Initially a constant fraction of all individuals had i expressed with sufficiently high probability (see Lemma 2.6). Each round of crossover works like the pull protocol in rumor spreading to get the variable i expressed, so that Lemma 5.4 gives that, after $O(\log\log n)$ rounds of crossover, all individuals have i expressed with probability at least $1-n^{-6}$.

While the size of a tree is at most n^2, it grows at most by a factor of $2\log n$, leading to a tree size of $n(\log n)^{O(\log\log n)} \leq n^{1+o(1)}$ after $O(\log\log n)$ iterations.

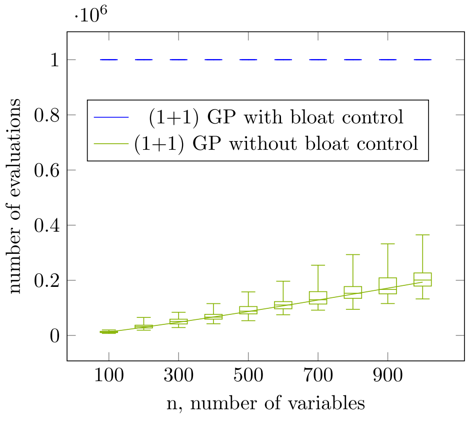
In order to compute the total run time of all local searches, note that the growth of individuals is faster than exponential, and since we only desire an asymptotic bound we can ignore all but the last summand. We apply local search to each of logarithmically many individuals, which increases the run time by another logarithmic factor, which still gives a bound of $n^{1+o(1)}$. A union bound over all failure probabilities gives the desired result. □

6. Experiments

This section is dedicated to complementing our theoretical results with experimental justification for the otherwise open cells of Table 1, i.e. for the (1+1) GP without bloat control on $+c$-Majority and 2/3-Majority.

All experimental results shown in Fig. 4 are box-and-whiskers plots, where lower and upper whiskers are the minimal and maximal number of *fitness evaluations* the algorithm required over 100 runs until all variables are expressed or the time limit of 1000000 evaluations is reached. The middle lines in each box are the median values (the second quartile), the bottom and top of the boxes are the first and third quartiles. Note that all experiments are platform independent since we count number of fitness evaluations independently of real time. The solid lines in the plots allow to estimate the asymptotic run time of the (1+1) GP.

The left hand side of Fig. 4 concerns $+c$-Majority and shows that the (1+1) GP with bloat control always fails (corresponding to Theorem 3.1). We used the (1+1) GP with $s_{\text{init}} = 10n$, $c = 2$ and n as indicated along the x-axis. It is easy to see that bloat control leads the algorithm to local optima and does not allow to leave it, whereas the (1+1) GP *without* bloat control finds an optimum in a reasonable number of evaluations. Due to time and computational restrictions the constant c was chosen equal to 2. For larger c the run time of the algorithm goes up significantly, but a similar pattern is visible.



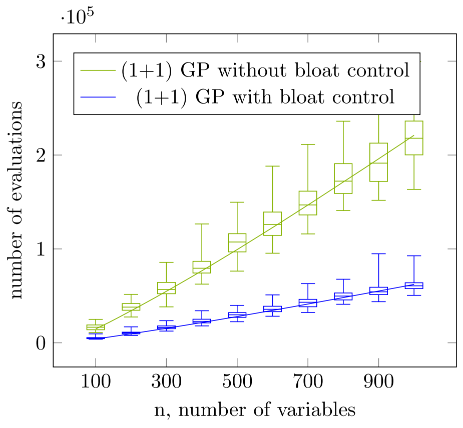


Fig. 4. Number of evaluations required by the (1+1) GP over 100 runs for each n with the initial tree size $s_{\text{init}} = 10n$ until all variables are expressed or the time limit, equal to 1000000 evaluations, is reached. The left figure shows the experimental results for $+c - $ Majority with $c = 2$; the solid line is $28n \log n$. Note that all experiments with bloat control timed out, so there are no boxes and whiskers to be shown. On the right figure is shown $2/3 -$ Majority; the blue solid line is $9n \log n$, the green solid line is $32n \log n$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The right hand side of Fig. 4 shows the results of (1+1) GP on 2/3-Majority, using $s_{\text{init}} = 10n$. One can see that bloat control is more efficient in comparison with the (1+1) GP without bloat control. The set of median values is well-approximated by $w \cdot n \log n$ for a constant w, which leads us to the conjecture that the algorithm's run time is $O(n \log n)$. We did not analyze the influence of s_{init}, but it might be significant especially for 2/3-Majority without bloat control.

The source code we used can be found at https://github.com/melnan/1-1-GP.

7. Conclusion

We defined three variants of the Majority problem in order to introduce some fitness plateaus that are difficult to cross. The $+c$-Majority allows for progress at the end of the plateau with large representation; in this sense, bloat is necessary for progress. On the other hand, for 2/3-Majority, progress can be made at the end of the plateau with small representation, so that bloat control guides the search to the fruitful part of the search space. We also considered 2/3-SuperMajority which exemplifies fitness functions where bloat is inherent due to the possibility of small improvements by adding an increasing amount of nodes to the GP-tree. In this case we showed that not employing bloat control leads to inefficient optimization.

In order to obtain results somewhat closer to practically relevant GP we turned to crossover and showed how a Concatenation Crossover GP, employing bloat control, can efficiently optimize all three considered test functions. However, we believe that there is still a lot to be done before insights for improved practical GP algorithms can be derived. We need analyses of more realistic settings where the tree structure matters, as well as more relevant crossover operators. This might require the development of additional test functions, making essential use of the tree structure (all our test functions might as well use lists or even multisets of the leaves as representations). Such test functions should not be too complex, which would hinder a theoretical analysis, but still embody a structure frequently found in GP, so as to inform about relevant application areas. The search for such test functions remains a central open problem of the theory of GP.

Declaration of competing interest

The authors have no conflict of interest with respect to this manuscript.

Acknowledgements

We thank the COST Action CA15140 on nature-inspired search heuristic for inspiration.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Robert L. Rivest, Clifford Stein, Introduction to Algorithms, 2 edition, MIT Press, 2001.
[2] Benjamin Doerr, Leslie Ann Goldberg, Adaptive drift analysis, Algorithmica 65 (1) (2013) 224–250.
[3] Benjamin Doerr, Timo Kötzing, J.A. Gregor Lagodzinski, Johannes Lengler, Bounding bloat in genetic programming, in: Proc. of GECCO'17, 2017, pp. 921–928.
[4] Benjamin Doerr, Andrei Lissovoi, Pietro Simone Oliveto, Evolving Boolean functions with conjunctions and disjunctions via genetic programming, CoRR, arXiv:1903.11936, 2019.
[5] Devdatt P. Dubhashi, Alessandro Panconesi, Concentration of Measure for the Analysis of Randomized Algorithms, Cambridge University Press, 2009.
[6] Greg Durrett, Frank Neumann, Una-May O'Reilly, Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics, in: Proc. of FOGA'11, 2011, pp. 69–80.

[7] Brent E. Eskridge, Dean F. Hougen, Memetic crossover for genetic programming: evolution through imitation, in: Proc. of GECCO'04, 2004, pp. 459–470.
[8] Chris Gathercole, Peter Ross, An adverse interaction between the crossover operator and a restriction on tree depth, in: Proc. of GP'96, 1996, pp. 291–296.
[9] David E. Goldberg, Una-May O'Reilly, Where does the good stuff go, and why? How contextual semantics influences program structure in simple genetic programming, in: Proc. of EuroGP'98, 1998, pp. 16–36.
[10] Jun He, Xin Yao, A study of drift analysis for estimating computation time of evolutionary algorithms, Nat. Comput. 3 (1) (2004) 21–35.
[11] Daniel Johannsen, Random Combinatorial Structures and Randomized Search Heuristics, PhD thesis, Universität des Saarlandes, 2010, Available online at http://scidok.sulb.uni-saarland.de/volltexte/2011/3529/pdf/Dissertation_3166_Joha_Dani_2010.pdf.
[12] Terry Jones, Crossover, macromutation, and population-based search, in: Proc. of ICGA'95, 1995, pp. 73–80.
[13] Terry Jones, Evolutionary Algorithms, Fitness Landscape and Search, PhD thesis, University of New, Mexico, 1995.
[14] Richard M. Karp, Christian Schindelhauer, Scott Shenker, Berthold Vöcking, Randomized rumor spreading, in: Proc. of FOCS'00, 2000, pp. 565–574.
[15] Timo Kötzing, Frank Neumann, Reto Spöhel, PAC learning and genetic programming, in: Proc. of GECCO'11, 2011, pp. 2091–2096.
[16] Timo Kötzing, Andrew M. Sutton, Frank Neumann, Una-May O'Reilly, The Max problem revisited: the importance of mutation in genetic programming, in: Proc. of GECCO'12, 2012, pp. 1333–1340.
[17] John R. Koza, Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, Technical report, Stanford, CA, USA, 1990.
[18] Timo Kötzing, Concentration of first hitting times under additive drift, Algorithmica 75 (3) (2016) 490–506.
[19] William B. Langdon, Riccardo Poli, An analysis of the MAX problem in genetic programming, in: Proc. of GP'97, 1997, pp. 222–230.
[20] Andrei Lissovoi, Pietro Simone Oliveto, On the time and space complexity of genetic programming for evolving Boolean conjunctions, in: Proc. of AAAI'18, 2018, pp. 1363–1370.
[21] Sean Luke, Liviu Panait, Lexicographic parsimony pressure, in: Proc. of GECCO'02, 2002, pp. 829–836.
[22] Sean Luke, Liviu Panait, A comparison of bloat control methods for genetic programming, Evol. Comput. 14 (2006) 309–344.
[23] Andrea Mambrini, Luca Manzoni, A comparison between geometric semantic GP and Cartesian GP for Boolean functions learning, in: Proc. of GECCO'14, 2014, pp. 143–144.
[24] Andrea Mambrini, Pietro Simone Oliveto, On the analysis of simple genetic programming for evolving Boolean functions, in: Proc. of EuroGP'16, 2016, pp. 99–114.
[25] Hugues Mercier, Laurent Hayez, Miguel Matos, Optimal epidemic dissemination, CoRR, arXiv:1709.00198, 2017.
[26] Michael Mitzenmacher, Eli Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis, Cambridge University Press, New York, NY, USA, 2005.
[27] Alberto Moraglio, Krzysztof Krawiec, Colin G. Johnson, Geometric semantic genetic programming, in: Proc. of PPSN'12, Springer, 2012, pp. 21–31.
[28] Alberto Moraglio, Andrea Mambrini, Luca Manzoni, Runtime analysis of mutation-based geometric semantic genetic programming on Boolean functions, in: Proc. of FOGA'13, 2013, pp. 119–132.
[29] Frank Neumann, Computational complexity analysis of multi-objective genetic programming, in: Proc. of GECCO'12, 2012, pp. 799–806.
[30] Anh Nguyen, Tommaso Urli, Markus Wagner, Single- and multi-objective genetic programming: new bounds for weighted ORDER and MAJORITY, in: Proc. of FOGA'13, 2013, pp. 161–172.
[31] Pietro S. Oliveto, Carsten Witt, Simplified drift analysis for proving lower bounds in evolutionary computation, Algorithmica 59 (3) (2011) 369–386.
[32] Pietro S. Oliveto, Carsten Witt, Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation, e-prints, http://arxiv.org/abs/1211.7184, 2012.
[33] Una-May O'Reilly, An Analysis of Genetic Programming, PhD thesis, Carleton University, Ottawa, Canada, 1995.
[34] Una-May O'Reilly, Franz Oppacher, Program search with a hierarchical variable length representation: genetic programming, simulated annealing and hill climbing, in: Proc. of PPSN'94, 1994, pp. 397–406.
[35] Markus Wagner, Frank Neumann, Parsimony pressure versus multi-objective optimization for variable length representations, in: Proc. of PPSN'12, 2012, pp. 133–142.