Improved Computational Complexity Results for Weighted ORDER and MAJORITY

Anh Nguyen
Evolutionary Computation Group
School of Computer Science
The University of Adelaide
Adelaide, SA 5005, Australia

Tommaso Urli
Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica
Università degli Studi di Udine
33100 Udine, Italy

Markus Wagner
Evolutionary Computation Group
School of Computer Science
The University of Adelaide
Adelaide, SA 5005, Australia

ABSTRACT

We consolidate the existing computational complexity analysis of genetic programming (GP) by bringing together sound theoretical proofs and empirical analysis. In particular, we address computational complexity issues arising when coupling algorithms using variable length representation, such as GP itself, with different bloat-control techniques. In order to accomplish this, we first introduce several novel upper bounds for two single- and multi-objective GP algorithms on the generalised Weighted ORDER and MAJORITY problems. To obtain these, we employ well-established computational complexity analysis techniques such as fitness-based partitions, and for the first time, additive and multiplicative drift.

The bounds we identify depend on two measures, the maximum tree size and the maximum population size, that arise during the optimization run and that have a key relevance in determining the runtime of the studied GP algorithms. In order to understand the impact of these measures on a typical run, we study their magnitude experimentally, and we discuss the obtained findings.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Theory, Algorithms, Performance

Keywords

Genetic Programming, Multi-objective Optimization, Theory, Runtime Analysis

1. INTRODUCTION

In the last decade, genetic programming (GP) has found various applications (see Poli et al., 2008) in a number of domains. As other paradigms based on variable length representation, however, GP can be subject to *bloating*. Bloating occurs when a solution's growth in complexity does not correspond to a growth in quality and causes the optimization process to diverge and slow down. Since bloat-control is a key factor to the efficient functioning of GP, its impact on the computational complexity has been studied already for simple problems, in Durrett et al. (2011) and Neumann (2012). The algorithms that have been considered are a stochastic hill-climber called (1+1)-GP, and a population-based multi-objective genetic programming algorithm called SMO-GP; the latter considers the trade-offs between solutions complexity C and fitness with respect to a problem F. These algorithms have been analysed on problems with isolated program semantics taken from Goldberg and O'Reilly (1998), namely ORDER and MAJORITY, which can be seen as the analogue of linear pseudo-Boolean functions Droste et al. (2002) that are known from the computational complexity analysis of evolutionary algorithms working with fixed length binary representations. Both problems are simple enough to be analysed thoroughly, and they represent different aspects of problems solved through genetic programming, that is, including components in the correct order (ORDER), and including the correct set of components in a solution (MAJORITY). Additional recent computational complexity results are those of Kötzing et al. (2012) on the MAX problem, and of Wagner and Neumann (2012) on the SORTING problem.

The results provided in Durrett et al. (2011); Neumann (2012) raise several questions that remain unanswered in both papers. In particular, for different combinations of algorithms and problems no (or no exact) runtime bounds are given. These works suggest that two measures, namely the maximum tree size T_{max} and the maximum population size P_{max} obtained during the run, play a role in determining the expected optimization time of the investigated algorithms. Urli et al. (2012) have made significant effort to study the order of growth of these quantities, and to conjecture runtime bounds for both problems. However, their results are based purely on extensive experimental investigations, potentially neglecting problematic cases. Even though Urli et al. (2012) conjecture runtimes based on their observations, the impact of both quantities on the runtime is still unclear from the theory side.

In this paper, we address these questions. We use multiplicative drift (Doerr et al., 2010) on the fitness values to bound the runtime of (1+1)-GP on the *Weighted* ORDER (WORDER) problem. Subsequently, we consider (1+1)-GP on the multi-objective formulations of WORDER, which considers the complexity as well. There, we apply drift analysis on the solution sizes in order to bound (with high probability) the maximum tree sizes encountered. Lastly, we consider the multi-objective SMO-GP algorithm, and bound the runtimes using fitness-based partitions. In the cases where T_{max} and P_{max} are part of the asymptotic bound, we augment the results with experimental observations.

Note that our investigations focus on the weighted variants WORDER and WMAJORITY, which both allow for exponentially many different fitness values. Very few runtime bounds were known for both problems so far.

The paper is structured as follows. In Section 2, we introduce the analysed problems and algorithms. In Section 3, we summarize the previous computational complexity results from Durrett et al. (2011); Neumann (2012). In Sections 4 and 5, we present several new theoretical upper bounds and we complement the analyses with experimental results whenever a term of the bound is not under the control of the user. In the final section, we summarise the existing known bounds and ours, and we point out open questions.

2. PRELIMINARIES

In our theoretical and experimental investigations, we will treat the algorithms and problems analyzed in Durrett et al. (2011); Neumann (2012). We consider tree-based genetic programming where a possible solution is represented by a syntax tree. The inner nodes of such a tree are labelled by function symbols from a set F and the leaves of the tree are labelled by terminals from a set T.

We examine the problems Weighted ORDER (WORDER) and Weighted MAJORITY (WMAJORITY). In these problems, the only function symbol is the join (denoted by J), which is binary. The terminal set is a set of $2n$ variables, where $\bar{x}_i$ is considered the complement of x_i. Hence, $F := \{J\}$, and $L := \{x_1, \bar{x}_1, x_2, \bar{x}_2, ..., x_n, \bar{x}_n\}$.

In WORDER and WMAJORITY, each variable x_i is assigned a weight $w_i \in \mathbb{R}$, $1 \leq i \leq n$ so that the variables can differ in their contributions to the fitness of a tree. Without loss of generality, we assume that $w_1 \geq w_2 \geq w_3 \geq \ldots \geq w_n > 0$. We get the ORDER and MAJORITY as specific cases of WORDER and WMAJORITY where $w_i = 1$, $1 \leq i \leq n$.

For a given solution X, the fitness value is computed by parsing the represented tree inorder. For WORDER, the weight w_i of a variable x_i contributes to the fitness of X iff x_i is visited in the inorder parse before all the $\bar{x}_i$ in the tree. For WMAJORITY, the weight of x_i contributes to the fitness of X iff the number of occurrences of x_i in the tree is at least one and not less than the number of occurrence of $\bar{x}_i$ (see Figures 2 and 3). We call a variable redundant if it occurs multiple times in the tree; in this case the variable contributes only once to the fitness value. The goal of WORDER and WMAJORITY problems is to maximize their function values. We illustrate both problems by an example (see Figure 1).

MO-WORDER and MO-MAJORITY are variants of the above-described problems, which take the complexity C of a syntax tree (computed by the number of leaves of the tree) as the second objective:

- MO-WORDER (X) = (WORDER (X), C(X))
- MO-WMAJORITY (X) = (WMAJORITY (X), C(X))

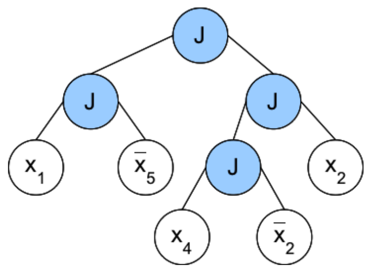

Figure 1: Example for evaluations according to WORDER and WMAJORITY. Let $n = 5$ and $w_1 = 15$, $w_2 = 14$, $w_3 = 12$, $w_4 = 7$, and $w_5 = 2$. For the shown tree X, we get (after inorder parsing) $l = (x_1, \bar{x_5}, x_4, \bar{x_2}, x_2)$. For WORDER, we get $S = (x_1, x_4)$ and WORDER$(X) = w_1 + w_4 = 22$. For WMAJORITY, we get $S = (x_1, x_4, x_2)$ and WMAJORITY$(X) = w_1 + w_4 + w_2 = 36$.

```
Input: a syntax tree $X$
Init: $l$ an empty leaf list, $S$ is an empty statement list.
  1. Parse $X$ inorder and insert each leaf at the rear of $l$ as it
     is visited.
  2. Generate $S$ by parsing $l$ front to rear and adding ("ex-
     pressing") a leaf to $S$ only if it or its complement are
     not yet in $S$ (i.e. have not yet been expressed).
  3. WORDER (X) = $\sum_{x_i \in S} w_i$
```

Figure 2: Computation of WORDER (X)

Optimization algorithms can then use this to cope with the bloat problem: if two solutions have the same fitness value, then the solution of lower complexity can be preferred. In the special case, where $w_i = 1$ holds for all $1 \leq i \leq n$, we have the problems:

- MO-ORDER (X) = (ORDER (X), C(X))
- MO-MAJORITY (X) = (MAJORITY (X), C(X))

In this paper, all algorithms only use the mutation operator HVL-Prime to generate offspring. HVL-Prime produces a new tree by making changes to the original tree via three basic operators: insertion, deletion and substitution. A more detailed explanation of this operator can be found in Durrett et al. (2011). In each step of the algorithms, k mutations are applied to the selected solution. For the single-operation variants of the algorithms, $k = 1$ holds. For the multi-operation variants, the number of operations performed

```
Input: a syntax tree $X$
Init: $l$ an empty leaf list, $S$ is an empty statement list.
  1. Parse $X$ inorder and insert each leaf at the rear of $l$ as it
     is visited.
  2. For $1 \leq i \leq n$, if count($x_i \in l$) $\geq 1$ and count($x_i \in l$) $\geq$
     count($\bar{x}_i \in l$), add $x_i$ to $S$
  3. WMAJORITY (X) = $\sum_{x_i \in S} w_i$
```

Figure 3: Computation of WMAJORITY (X)

Mutate Y by applying HVL-Prime k times, In each time, randomly choose either insert, subsitute or delete.

- Insert: Choose a variable $u \in L$ uniformly at random and select a node $v \in Y$ uniformly at random. Replace v by a join node whose children are u and v, in which their orders are chosen randomly,
- Substitute: Replace a randomly chosen leaf $v \in Y$ by a randomly chosen leaf $u \in L$.
- Delete: Choose a leaf node $v \in Y$ randomly with parent p and sibling u. Replace p by u and delete p and u.

Figure 4: HVL-Prime mutation operator

is drawn each time from the distribution $k = 1 + Pois(1)$, where $Pois(1)$ is the Poisson distribution with parameter 1.

3. THEORETICAL RESULTS

The computational complexity analysis of genetic programming analyses the expected number of fitness evaluations until the algorithm has produced an optimal solution for the first time. This is called the *expected optimization time*. In the case of multi-objective optimization the definition of expected optimization time is slightly different and considers the number of fitness evaluations until the whole Pareto front, i.e. the set of optimal trade-offs between the objectives, has been computed.

The existing bounds from Durrett et al. (2011); Neumann (2012) are listed in Table 1. As it can be seen, all results for (1+1)-GP take into account tree sizes of some kind: either the maximum tree size T_{max} found during search or the initial tree T_{init} size play a role in determining the bound. While T_{init} is something which can be decided in advance, T_{max} is a result of the interactions between the fitness function, the set of mutations and the selection process. As mutations involve a degree of randomness, in some cases such a measure is very difficult to control.

A similar problem arises when dealing with multi-objective algorithms, such as SMO-GP. As we will see, the maximum population size reached during optimization, P_{max}, is a fundamental term in most bounds and it is again very difficult to tackle when a random number of mutations is involved. This is the paramount reason for which the only bounds for the single-mutation variant are known to date. Lastly, note that the upper bounds marked with $\star$ hold only if the algorithm has been initialized in the particular, i.e. non-redundant, way described in Neumann (2012).

Because of the direct relation between these measures and the bounds, investigating their magnitude is key to the fundamental understanding of the bounds meaning.

4. (1+1)-GP

The algorithm (1+1)-GP starts with an initial solution X, and in each generation a new offspring Y is produced by mutating X. Y replaces X if it is favoured by the selection mechanism. Different from WORDER and WMAJORITY where solutions of equal fitnesses can replace each other unconditionally, the selection on MO-WORDER and MO-WMAJORITY favours solutions with higher fitness value or smaller complexity value when two solutions have the same fitness value.

Algorithm 1: (1+1)-GP algorithm

```
1. Choose an initial solution $X$
2. Repeat
     • Set $Y := X$
     • Apply mutation to $Y$
     • If the selection favours $Y$ over $X$ then $X := Y$.
```

Let X, Y be the solutions and F be the fitness function in (1+1)-GP.

- (1+1)-GP on *F*: Favour Y over X iff $F(Y) \geq F(X)$
- (1+1)-GP on *MO-F*: Favour Y over X iff $(F(Y) > F(X)) \vee ((F(Y) = F(X)) \wedge ((C(Y) \leq C(X)))$

Figure 5: Selection mechanism of (1+1)-GP

However, since in MO-WORDER and MO-WMAJORITY, the complexity of a syntax tree is not taken into account, there is no mechanism for handling the bloat problem. Given the maximum tree size T_{max} for a run is unknown, it would be preferable to have runtime bounds based on the size of the initial tree T_{init}, which the user can control. Using a different approach in which the selection in Figure 4 is used, Neumann (2012) proved that the expected optimisation time for (1+1)-GP-single on both MO-WORDER and MO-WMAJORITY is $O(T_{init} + n \log n)$. In the following subsections, (1+1)-GP with one and more than one mutation in each step are correspondingly denoted by (1+1)-GP-single and (1+1)-GP-multi.

4.1 (1+1)-GP on F(X) Problems

In previous works, Durrett et al. (2011) showed that the upper bound of the expected run time for (1+1)-GP on ORDER is $O(nT_{max})$. First, we show that this bound immediately carries over for (1+1)-GP-single on WORDER.

THEOREM 1. *The expected optimization time of (1+1)-GP-single on WORDER is* $O(nT_{max})$.

PROOF. As (1+1)-GP-single performs only a single mutation operation at a time, it is not possible (1) to increase the WORDER value of a tree and (2) to decrease the number of expressed variables at the same time. Hence, once a variable is expressed, it will stay expressed for the rest of the optimisation process. This, however, means that the weights associated to the variables are effectively irrelevant. Thus, the upper bound of the runtime is identical to that of (1+1)-GP-single on the unweighted ORDER, and the bound immediately carries over. □

The bound for the multi-operation variant is found by applying **Theorem 3** (multiplicative drift analysis) from Doerr et al. (2010), which is defined as follows.

F(X)	(1+1)-GP, F(X) Durrett et al. (2011)		(1+1)-GP, MO-F(X) Neumann (2012)		SMO-GP, MO-F(X) Neumann (2012)	
	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)
ORDER	$O(nT_{max})$	$O(nT_{max})$	$O(T_{init} + n \log n)$	?	$O(nT_{init} + n^2 \log n)$	
WORDER	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?
MAJORITY	$O(n^2 T_{max} \log n)$	?	$O(T_{init} + n \log n)$		$O(nT_{init} + n^2 \log n)$	
WMAJORITY	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?

Table 1: Computational complexity results from Durrett et al. (2011); Neumann (2012)

THEOREM 2 (MULT. DRIFT, DOERR ET AL. (2010)). *Let $S \subseteq \mathbb{R}$ be a finite set of positive numbers with minimum s_{min}. Let $X^{(t)}{}_{t \in \mathbb{N}}$ be a sequence of random variables over $S \cup \{0\}$. Let T be the random variable that denotes the first point in time $t \in \mathbb{N}$ for which $X^{(t)} = 0$.*

Suppose that there exists a constant $\delta > 0$ such that

$$E\left[X^{(t)} - X^{(t+1)} | X^{(t)} = s\right] \geq \delta s \quad (1)$$

holds for all $s \in S$ with $\Pr[X^t = s] > 0$. Then for all $s_0 \in S$ with $\Pr[X^{(0)} = s_0] > 0$,

$$E[T | X^{(0)} = s_0] \leq \frac{1 + \log(s_0/s_{min})}{\delta}. \quad (2)$$

In the following theorem we employ Theorem 2 to provide an upper bound on the expected runtime of (1+1)-GP-multi, F(X) on WORDER when starting from any solution.

THEOREM 3. *The expected optimization time of (1+1)-GP-multi on WORDER is $O(nT_{max}(\log n + \log w_{max}))$.*

PROOF. In order to prove the stated bounds we need to instantiate Theorem 2 on our problem. The theorem is valid if we can identify a finite set $S \subseteq \mathbb{R}$ from which the random variables X^t are drawn. In our case the drift is defined on the (at most exponentially many) values of the fitness function. Also, since the result holds under the assumption that the underlying problem is a minimization problem, we first need to define an auxiliary problem WORDER$_{min}$ which get minimized whenever WORDER gets maximized

$$\text{WORDER}_{min}(x) = \sum_{i=1}^{n} w_i - \text{WORDER}(x).$$

Note that while WORDER(x) is the sum of the weights for the expressed variables in x, WORDER$_{min}(x)$ is the sum of the weights of the unexpressed variables. We also recall that ORDER(x) denotes the number of expressed variables in x.

Now, given a solution x^t at time t, let $s_t = \text{WORDER}_{min}(x^t)$ be the WORDER-value of this solution and $m = n - \text{ORDER}(x^t)$ the number of unexpressed variables in x^t. Since an improvement can be done by inserting a non-expressed variables into the current tree, then the expected increment, i.e. the drift, of WORDER$_{min}$ is lower bounded by:

$$\begin{aligned} E\left[X^{(t)} - X^{(t+1)} | X^{(t)} = s_t\right] &\geq \frac{\text{WORDER}_{min}(x^t)}{m} \cdot \frac{m}{6enT_{max}} \\ &= \frac{\text{WORDER}_{min}(x^t)}{6enT_{max}} \end{aligned}$$

since the probability of expressing any of the m unexpressed variables at the beginning of the tree is at least $\frac{m}{2n} \cdot \frac{1}{3eT_{max}} = \frac{m}{6enT_{max}}$ for both the single- and the multi-operation case, and the expected improvement when performing such step is $WORDER_{min}(x^t)/m$ (because the missing weights are distributed over m variables).

Given this drift, we have (in Theorem 2 terminology)

$$\begin{aligned} \delta &= \frac{1}{6enT_{max}} \\ s &= \text{WORDER}_{min}(x^t). \end{aligned}$$

Also, from the definition of WORDER, we have that $s_{min} = w_{min}$. From Theorem 2 follows that, for any initial value $s_0 = \text{WORDER}_{min}(x^0)$, we have

$$\begin{aligned} E[T | X^{(0)} = s_0] &\leq \left(1 + \ln\left(\frac{s_0}{w_{min}}\right)\right) \cdot \delta^{-1} \\ &\leq \left(1 + \ln\left(\frac{s_0}{w_{min}}\right)\right) \cdot 6enT_{max}. \end{aligned}$$

Since we are interested in an upper bound over the expected optimization time, we must consider starting from the worst possible initial solution, i.e. where none of the variables is expressed. Let w_{max} be the largest weight in the set, then the maximum distance from the optimal solution is $nw_{max} \geq s_0$. Hence we have

$$\begin{aligned} E[T] &\leq \left(1 + \ln\left(\frac{nw_{max}}{w_{min}}\right)\right) \cdot 6enT_{max} \\ &\leq (1 + \ln(nw_{max})) \cdot 6enT_{max} \\ &= O(nT_{max} \log(nw_{max})) = O(nT_{max}(\log n + \log w_{max})) \end{aligned}$$

which states that the expected optimization time T of (1+1)-GP, F(X) for WORDER$_{min}$, and hence for WORDER, starting from any solution is bounded by $O(nT_{max}(\log n + \log w_{max}))$. □

The dependence of this bound, and of the bound for ORDER introduced in Durrett et al. (2011), on T_{max} is easily explained by the fact that, in order to guarantee a single-step improvement, one must perform a beneficial insertion, i.e. one which expresses one of the unexpressed variables, at the beginning of the tree. This involves selecting one out of at most T_{max} nodes in the tree. Unfortunately, T_{max} can potentially grow arbitrarily large since $F(X)$ does not control solution complexity.

We have investigated this measure experimentally in order to understand what its typical magnitude is. The experiments were performed on AMD Opteron 250 CPUs (2.4GHz), on Debian GNU/Linux 5.0.8, with Java SE RE 1.6 and were given a maximum runtime of 3 hours and a budget of 10^9 evaluations. Furthermore, each experiment (involving two different initialization schemes, respectively with 0 and $2n$ leaves in the initial tree) has been repeated 400 times, which results in a standard error of the mean of $1/\sqrt{400} = 5\%$. The empirical distributions of maximum tree sizes for (1+1)-GP, $F(X)$ on all ORDER variants are shown as box-plots in Figure 6. The blue-toned line plots show the median T_{max} divided by n (solid line) and by $n\log(n)$ (dashed line); the nearly constant behavior of the solid line suggests that T_{max} has typical a linear behavior, at least for the tested values of n and the employed initialization schemes, for all the variants of ORDER.

4.2 Algorithms on MO-F(X) Problems

For the multi-operation variants, a single HVL-Prime application can lead to more than a single mutation with constant probability. This would lead to the case where the complexity increase faster than the fitness value, i.e. an improvement on the fitness of the tree can be followed by multiple increase on the complexity. We start our analysis by showing an upper bound of the maximum tree size, T_{max}, during run time and then using that fact to bound the expected optimisation time of (1+1)-GP-multi on MO-WORDER.

THEOREM 4. *(Oliveto and Witt, 2011) Let $X_t, t \geq 0$, be the random variables describing a Markov process over a finite state space $S \subseteq \mathbb{R}_0^+$ and denote $\Delta_t(i) := (X_{t+1} - X_t \mid X_t = i)$ for $i \in S$ and $t \geq 0$. Suppose there exists an interval $[a,b]$ in the state space, two constants $\delta, \varepsilon > 0$ and, possibly depending on $l := b - a$, a function $r(l)$ satisfying $1 \leq r(l) = o(l/log(l))$ such that for all $t \geq 0$ the following two conditions hold:*

1. *$E(\Delta_t(i)) \geq \varepsilon$ for $a < i < b$,*
2. *$Prob(\Delta_t(i) \leq -j) \leq \frac{r(l)}{(1+\delta)^j}$ for $i > a$ and $j \in \mathbb{N}_0$.*

Then there is a constant $c^* > 0$ such that for $T^* := \min\{t \geq 0 : X_t \leq a \mid X_0 \geq b\}$ it holds $Prob\left(T^* \leq 2^{c^* l/r(l)}\right) = 2^{-\Omega(l/r(l))}$. In the conference version, $r(l)$ was only allowed to be a constant, i.e., $r(l) = O(1)$. In this case, the last statement is simplified to

$$Prob(T^* \leq 2^{c^* l}) = 2^{-\Omega(l)}.$$

THEOREM 5. *Let $T_{init} \leq 19n$ be the complexity of the initial solution. Then, the maximum tree size encountered by (1+1)-GP-multi on MO-WORDER in less than exponential time is $20n$, with high probability.*

Note that the state space S (of the tree sizes) is finite for both problems, as at most 2^n fitness improvements are accepted. Thus, at most $2^n + 1$ different tree sizes can be attained.

PROOF. Theorem 4 was used to find the lower bound value of a function in less than exponential time. In case of finding the upper bound value of a function, condition 1 and 2 of Theorem 4 are then changed to :

1. $E(\Delta_t(i)) \leq -\varepsilon$ for $a < i < b$ and $\varepsilon > 0$,
2. $Prob(\Delta_t(i) \geq j) \leq \frac{r(l)}{(1+\delta)^j}$ for $i > a$ and $j \in \mathbb{N}_0$.

Let $a = 19n$, $b = 20n$, k be the number of expressed variables, and s be the number of leaves of the current tree. For condition 1 to hold, the expected drift of the size of a syntax tree in the interval $[a,b] = [19n, 20n]$ must be a negative constant. This is computed by

$$\begin{aligned}
E(\Delta_t(i)) &= \sum_{j\in\mathbb{Z}} j \cdot P(\Delta_t(i) = j) \\
&= \sum_{j\in\mathbb{Z}, j<0} j \cdot P(\Delta_t(i) = j) + \sum_{j\in\mathbb{N}} j \cdot P(\Delta_t(i) = j) \\
&= \sum_{j\in\mathbb{N}} -j \cdot P(\Delta_t(i) = -j) + \sum_{j\in\mathbb{N}} j \cdot P(\Delta_t(i) = j)
\end{aligned}$$

When $\Delta_t(i) = -j$ for $j \in \mathbb{N}$, then the tree size is reduced by j. We can find a lower bound to the expected decrease in size by observing that to obtain a reduction of $-j$, we can do in principle $t > j$ operations, out of which j must be deletions of redundant variables and the other $t - j$ must be neutral moves. Since we have three mutations with equal probability of being selected, there are at least to 3^{t-j} possible combinations of mutations that lead to a decrease of $-j$. We know that among these, at least one is made of substitutions that don't decrease the fitness and the probability of obtaining it $1/3^{t-j}$. Let $s = cn$, $19 < c < 20$, then

$$\begin{aligned}
&\sum_{j\in\mathbb{N}} -j \cdot P(\Delta_t(i) = -j) \\
&\leq -\sum_{j\in\mathbb{N}} j \sum_{t=j}^{\infty} \left[\left(\frac{1}{3}\right)^j \binom{s-k}{j} \left(\frac{1}{s}\right)^j \cdot \frac{1}{3^{t-s}} \cdot Pois(x = t) \right] \\
&\leq -\sum_{j\in\mathbb{N}} j \sum_{t=j}^{\infty} \left[\left(\frac{1}{3}\right)^j \binom{s-k}{j} \left(\frac{1}{s}\right)^j \cdot \frac{1}{3^{t-s}} \cdot \frac{1}{et!} \right] \\
&\leq -\tfrac{1}{e} \sum_{j\in\mathbb{N}} j \left(\frac{1}{3}\right)^j \cdot \left(\frac{s-k}{j}\right)^j \cdot \left(\frac{1}{s}\right)^j \sum_{t=j}^{\infty} \left(\frac{1}{3^{t-s}} \cdot \frac{1}{t!} \right) \\
&\leq -\tfrac{1}{e} \sum_{j\in\mathbb{N}} j \left(\frac{1}{3}\right)^j \cdot \left(\frac{c-1}{c}\right)^j \cdot \left(\frac{1}{j}\right)^j \sum_{t=j}^{\infty} \left(\frac{1}{3^{t-s}} \cdot \frac{1}{t!} \right)
\end{aligned}$$

where $(1/3)^j$ comes from the fact that we need to perform j deletions, $(1/s)^j$ is related to selecting the right leaf over s leaves for j times and $\binom{s-k}{j}$ is the number of possible permutations of redundant variables.

When $\Delta_t(i) = j$ for $j \in \mathbb{N}$, the size of the tree increases. In order to provide a upper bound on the expected increase of the tree size, we need to consider all the possible combinations of t mutations which lead us to an increase of j. Since we need to increase the tree size by j, we need at least j insertions, thus the remaining $t - j$ mutations can be arranged in 3^{t-j} different ways. Note that this is an upper bound, as many of these combinations will not lead to an increase of j. This means that the probability to do a correct number of

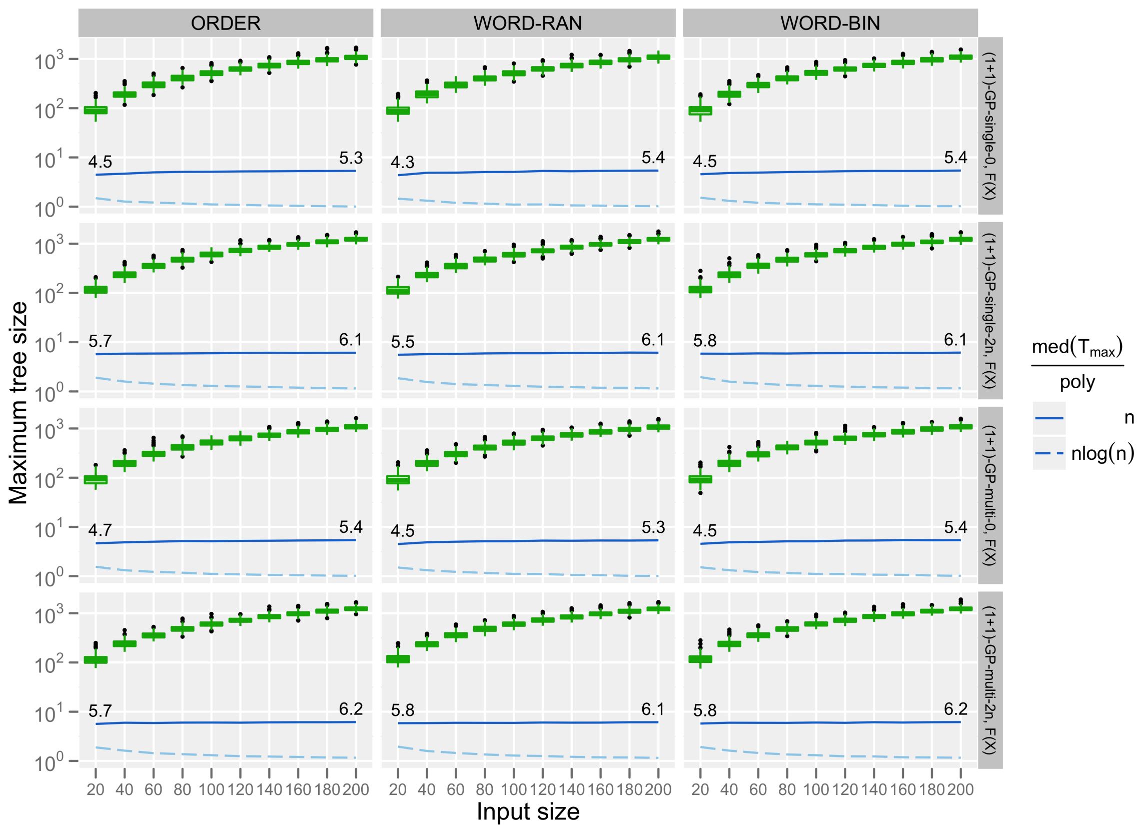

Figure 6: Distribution, as green box-plots, of the maximum tree sizes observed for (1+1)-GP, $F(X)$ on the ORDER variants. The blue-toned lines are the median T_{max} values divided by the corresponding polynomial (see the legend) and suggest the asymptotic behavior of the measure. In this case the almost horizontal line for n shows that the maximum tree size is very close to linear, at least for the tested input sizes.

insertions is upper bounded by

$$\frac{3^{t-j}}{3^t} = \frac{1}{3^j}$$

Also, in order to accept a larger tree size, the fitness must be increased as well. In the worst case, this can be accomplished by inserting one unexpressed variable in such a position that it increases the fitness of the tree size, an inserting $m-1$ random variables in positions that don't decrease the fitness of the tree. Thus the expected increase in the tree size is

$$\begin{aligned}
&\sum_{j\in\mathbb{N}} j \cdot P(\Delta_t(i) = j) \\
&\le \sum_{j\in\mathbb{N}} \left(j \cdot \frac{n-k}{2n} \cdot P_1 \cdot P_2 \sum_{t=j} Pois(x=t) \right) \\
&\le \sum_{j\in\mathbb{N}} \left(j \cdot \frac{1}{3^j} \cdot \frac{1}{2} \sum_{t=j} \frac{1}{et!} \right) \\
&= \frac{1}{e} \sum_{j\in\mathbb{N}} \left(j \cdot \frac{1}{2} \cdot \frac{1}{3^j} \sum_{t=j} \frac{1}{t!} \right)
\end{aligned}$$

where the $(1/3)^j$ term comes from the fact that we want to do j insertions, $(n-k)/2n$ comes from the fact that we want to insert one of the unexpressed variables, P_1 comes from the fact that we want to insert the unexpressed variable in a position where it can determine an increase in the fitness, and $P_2^{(j-1)}$ comes from the fact that we want to put $j-1$ (possibly redundant) random variables (whose probability 1 of being selected has been omitted) at any position in which they don't determine a decrease in the fitness. Also note that $P_1, P_2 \le 1$ and $n - k/2n \le 1/2$. Therefore, we have that

$$\begin{aligned}
E(\Delta_t(i)) = \quad & \sum_{j\in\mathbb{N}} -j \cdot P(\Delta_t(i) = -j) + \sum_{j\in\mathbb{N}} j \cdot P(\Delta_t(i) = j) \\
\le \quad & -\frac{1}{e} \sum_{j\in\mathbb{N}} \left[\frac{j}{j!} \cdot \left(\frac{1}{3}\right)^j \cdot \left(\frac{1}{cn}\right)^j \cdot \left(\frac{(c-1)n}{j}\right)^j \cdot \right. \\
& \left. \sum_{t=j}^{\infty} \left(\frac{1}{3^{t-s}} \cdot \frac{1}{t!} \right) \right] + \sum_{j\in\mathbb{N}} \left(j \cdot \frac{1}{2} \cdot \frac{1}{3^j} \cdot \frac{1}{e} \sum_{t=j} \frac{1}{t!} \right) \\
= \quad & \frac{1}{e} \sum_{j\in\mathbb{N}} \left(\frac{j}{3^j} \cdot \left[\frac{1}{2} \left(\sum_{t=j} \frac{1}{t!} \right) - \frac{1}{j!} \cdot \left(\frac{c-1}{c}\right)^j \cdot \right. \right. \\
& \left. \left. \frac{1}{j^j} \left(\frac{1}{j!} \frac{1}{3 \cdot (j+1)!} \right) \right] \right)
\end{aligned}$$

Let $A = \sum_{j\in\mathbb{N}} A_j$, where

$A_j = \frac{j}{3^j} \cdot \left[\frac{1}{2}\sum_{t=j}\frac{1}{t!} - \left(\frac{c-1}{c}\right)^j \frac{1}{j^j}\sum_{t=j}\left(\frac{1}{3^{t-j}}\frac{1}{t!}\right)\right]$

in which $\sum_{t=j}\left(\frac{1}{3^{t-j}}\frac{1}{t!}\right) \geq \left(\frac{1}{j!}\frac{1}{3\cdot(j+1)!}\right)$

for $19 < c < 20$, we have

$$A_1 < \frac{1}{3} \cdot \left(\frac{e-1}{2} - \frac{19}{20}(1+1/6)\right) = -0.083064$$
$$A_2 < \frac{2}{9} \cdot \left(\frac{e-2}{2} - \left(\frac{19}{20}\right)^2 \cdot \frac{1}{4}(1/2+1/18)\right) = 0.051954$$
$$A_3 < \frac{1}{9} \cdot \left(\frac{e-2-0.5}{2} - \left(\frac{19}{20}\right)^3 \cdot \frac{1}{27}(1/6+1/72)\right) = 0.011490$$

Hereby,

$$A = A_1 + A_2 + A_3 + \sum_{j\geq 4} A_j < -0.019620 + \sum_{j\geq 4} A_j$$

However, since

$$\begin{aligned}\sum_{j\geq 4} A_j &= \sum_{j\geq 4}\frac{j}{3^j} \cdot \left[\frac{1}{2}\sum_{t=j}\frac{1}{t!} - \left(\frac{c-1}{c}\right)^j \frac{1}{j^j}\sum_{t=j}\left(\frac{1}{3^{t-j}}\frac{1}{t!}\right)\right] \\ &< \sum_{j\geq 4}\frac{j}{3^j} \cdot \left(\frac{1}{2}\sum_{t=j}\frac{1}{t!}\right) \\ &< \sum_{j\geq 4}\frac{j}{3^j} \cdot \frac{e-2-0.5-1/6}{2} \\ &< 0.025808 \cdot \sum_{j\geq 4}\frac{j}{3^j} \\ &\leq 0.025808 \cdot \frac{1/3}{(1-1/3)^2} = 0.019356\end{aligned}$$

This gives us

$$\begin{aligned}A &= A_1 + A_2 + A_3 + \sum_{j\geq 4} A_j \\ &= -0.000264\end{aligned}$$

Therefore $E(\Delta_t(i)) = \frac{1}{e} \cdot A < 0$, and condition 1 holds.

For condition 2 to hold, in order to increase the number of nodes of the original tree by j, at least j insertions must be made. Thus, the probability that the tree size increases exactly by j is at most

$$\begin{aligned}P(\Delta_i = j) &\leq \frac{1}{e} \cdot \left(\frac{1}{2} \cdot \frac{1}{3^j}\sum_{t=j}\frac{1}{t!}\right) \\ &\leq \frac{1}{e} \cdot \frac{1}{3^j} \cdot \frac{1}{2} \cdot \sum_{t=j}\frac{1}{t!} \\ &\leq \frac{1}{3^j}\end{aligned}$$

Hence,

$$\begin{aligned}P(\Delta_i \geq j) &= \sum_{k=j}^{\infty} P(\Delta_i = k) \\ &\leq \sum_{k=j}^{\infty}\frac{1}{3^j} = \sum_{k=1}^{\infty}\frac{1}{3^j} - \sum_{k=1}^{j-1}\frac{1}{3^j} \\ &= \sum_{k=0}^{\infty}\frac{1}{3^j} - 1 - \left[\sum_{k=0}^{j-1}\frac{1}{3^j} - 1\right] \\ &= \frac{1}{1-1/3} - \frac{1-(1/3)^j}{1-1/3} = \frac{3}{2}\frac{1}{3^j}\end{aligned}$$

Let $r(l)$ be a constant and $\delta = 2$, then condition 2 holds. □

THEOREM 6. *Starting with an solution with initial size $T_{init} < 19n$, the optimization time of (1+1)-GP-multi on MO-ORDER is $O(n^2 \log n)$, with high probability.*

PROOF. This algorithm contains multiple phases. At each phase, there are two main tasks:

- Delete all the redundant variables in the current tree to obtain a non-redundant tree.
- Insert an unexpressed variable into the tree to improve the fitness

The algorithm terminates when all of n variables are expressed and the tree size is n. Let k_i, j_i be the number of the redundant and expressed variables of the solution at the beginning of phase i. The expected time T_i to delete all of the redundant variables at phase i is upper bounded by

$$\begin{aligned}T_i &= \sum_{l=1}^{k_i}(\frac{1}{3e})^{-1} \cdot (\frac{l}{l+j_i})^{-1} \leq 3e\sum_{l=1}^{k_i}\frac{l+n}{l} \\ &\leq 3e\sum_{l=1}^{n}\frac{l+n}{l} + 3e\sum_{l=n+1}^{k_i}\frac{l+n}{l} \\ &\leq 3e\sum_{l=1}^{n}\frac{l+n}{l} + 3e\sum_{l=n+1}^{k_i} 2 \\ &\leq O(n\log n) + 2k_i\end{aligned}$$

An unexpressed variable is then chosen and inserted in to the tree in any position to improve the fitness. The expected time to do such a step is $\frac{n-j_i}{6en}$. Therefore, the expected time for a single phase is $O(n\log n) + 2k_i + \frac{n-j_i}{6en}$.

The selection mechanism of (1+1)-GP on MO-F(X) problems does not accept any decrease in the fitness of the solution. Thus, j_i increases during the run time, which we can denote as $j_0 = 0 < j_1 = 1 < ... < j_{n-1} = n-1$. Since there are at most n variables that need to be inserted, there are at most n phases need to be done to obtain the optimal tree. The total runtime is then, with high probability, bounded by:

$$\begin{aligned}
&\sum_{j=0}^{n-1} O(n\log n) + 2k_i + \left(\frac{n-j}{6en}\right)^{-1} \\
&\leq O(n^2\log n) + 2\sum_{j=0}^{n-1} E\left[T_{max}\right] + 6en\sum_{j=0}^{n-1}\frac{i}{n-j} \\
&\leq O(n^2\log n) + 40n^2 + 6enO(\log n) \\
&\leq O(n^2\log n)
\end{aligned}$$

After n phases, the current tree now has the highest fitness value. The last step now is to remove all of the redundant variables to obtain a tree with highest fitness and complexity value of n. Let s be the number of leaves, the expected time for this step is upper bounded by:

$$\begin{aligned}
&\sum_{s=n+1}^{4n}\left(\frac{1}{3e}\cdot\frac{s-n}{s}\right)^{-1} = 3e\sum_{j=1}^{3n}\frac{j+n}{j} \\
&= 3e\sum_{j=1}^{n}\frac{j+n}{j} + 3e\sum_{j=n+1}^{2n}\frac{j+n}{j} + 3e\sum_{j=2n+1}^{3n}\frac{j+n}{j} \\
&\leq 3e\cdot\sum_{j=1}^{n}\frac{j+n}{j} + 3e\cdot\sum_{j=n+1}^{2n}2 + 3e\sum_{j=2n+1}^{3n}\frac{3}{2} \\
&= O(n\log n) + O(n)
\end{aligned}$$

Summing up the runtimes for all the tasks, the optimization time of (1+1)-GP-multi on MO-ORDER is $O(n^2\log n)$ with high probability. □

5. MULTI-OBJECTIVE ALGORITHMS

In SMO-GP, the complexity $C(X)$ of a solution X is also taken into account and treated as equally important as the fitness value $F(X)$. The classical Pareto dominance relations are:

1. A solution X *weakly dominates* a solution Y (denoted by $X \succeq Y$) iff $(F(X) \geq F(Y) \wedge C(X) \leq C(Y))$.
2. A solution X *dominates* a solution Y (denoted by $X \succ Y$) iff $((X \succeq Y) \wedge (F(X) > F(Y) \vee C(X) < C(Y))$.
3. Two solutions X and Y are called *incomparable* iff neither $X \succeq Y$ nor $Y \succeq X$ holds.

A *Pareto optimal solution* is a solution that is not dominated by any other solution in the search space. All Pareto optimal solutions together form the Pareto optimal set, and the set of corresponding objective vectors forms the Pareto front. The classical goal in multi-objective optimization (here: of SMO-GP) is to compute for each objective vector of the Pareto front a Pareto optimal solution. Alternatively, if the Pareto front is too large, the goal then is to find a representative subset of the front, where the definition of 'representative' depends on the choice of the conductor.

SMO-GP starts with a single solution, and at all times keeps a population that contains only the non-dominated solutions among the set of solutions seen so far.

Algorithm 2: SMO-GP

```
1. Choose an initial solution $X$
2. Set $P := \{X\}$
3. repeat
    • Randomly choose $X \in P$
    • Set $Y := X$
    • Apply mutation to $Y$
    • If $\{Z \in P | Z \succ Y\} = \varnothing$ set
      $P := (P \backslash \{Z \in P | Y \succ Z\}) \cup \{Y\}$
```

In his paper, Neumann (2012) proved that the expected optimization time of SMO-GP-single and SMO-GP-multi on MO-ORDER and MO-MAJORITY is $O(nT_{init} + n^2\log n)$. In the case of WORDER and WMAJORITY, the proven runtime bound of $O(n^3)$ (for both problems) requires that the algorithm is initialised with a non-redundant solution.

5.1 SMO-GP-single

If the initial solution is generated randomly, it is not likely to be a non-redundant one. In order to generalise the proof, we start the analysis by bounding the maximum tree size T_{max} based on the initial tree size T_{init} for SMO-GP-single. Using this result, we re-calculate the expected optimization time of SMO-GP-single on both MO-WORDER and MO-WMAJORITY, when using an arbitrary initial solution of size T_{init}.

THEOREM 7. *Let T_{init} be the tree size of initial solution. Then the population size of SMO-GP-single on MO-WORDER and MO-WMAJORITY until the empty tree included in the population is upper bounded by $T_{init} + n$ during the run of the algorithm.*

PROOF. When two solutions X, Y have the same complexity, the solution with higher fitness dominates the other one. Therefore, in the population where no solution is dominated by any other solutions, all the solutions have different complexities. Without loss of generality, assume that a solution X is selected for mutation and Y is the new solution. Then there are five ways to generate Y:

1. inserting an unexpressed variable, and thus increasing the complexity of the new solution by 1,
2. deleting an expressed variable, and thus decreasing the complexity of the new solution by 1,
3. deleting a redundant variable, and thus decreasing the complexity of the new solution by 1. As $F(X) = F(Y) \wedge C(X) > C(Y)$, X is then replaced by Y in the population,
4. inserting a redundant variable to X, thus generating a new solution with the same fitness and higher complexity, which is not accepted by SMO-GP-single,
5. substituting a variable x_i in X by another variable x_j.

- If x_j is a non-redundant variable and $w_i < w_j$, $F(X) < F(Y) \wedge C(X) = C(Y)$, Y replaces X in the population
- If x_j is a non-redundant variable and $w_i > w_j$, $F(X) > F(Y) \wedge C(X) = C(Y)$, Y is discarded.
- If x_j is redundant variable, $F(X) > F(Y) \wedge C(X) = C(Y)$, Y is discarded.

As shown above, the number of redundant variables in the solution will not increase during the run of the algorithm. Let R_i be the number of redundant variables in solution i, then for all the solutions in the population holds $R_i \leq R_{init}$.

Let $X \in P$ be a solution in P and α_X be the number of expressed variables in X, then the complexity of X is $C(X) = R_X + \alpha_X$. Since $R_X \leq R_{init}$ and $\alpha_X \leq n$, the complexity of a solution in P is upper bounded by $R_{init} + n \leq T_{init} + n$.

The population size only increases when a new solution with different fitness is generated. This only happens when the complexity increases or decreases by 1. Because the highest complexity that a solution can reach is $T_{init} + n$, and because the empty tree can be in the population as well[1], the maximum size the population can reach is $T_{init} + n + 1$. □

LEMMA 1. *Starting with an arbitrary initial solution of size T_{init}, the expected time until the population of SMO-GP-single on MO-WORDER and MO-WMAJORITY contains the empty tree is $O(T_{init}^2 + nT_{init})$.*

PROOF. We will bound the time by repeated deletions of randomly chosen leaf nodes in the solutions of lowest complexity, until the empty tree has been reached. Let X be the solution in the population with the lowest complexity. The probability of choosing the lowest complexity solution X in the population is at least $\frac{1}{T_{init}+n+1}$, while the probability of making a deletion is $\frac{1}{3}$. In each step, because any variable in X can be deleted, the probability of choosing the correct variable is 1. The probability of deleting a variable in X at each step therefore is bounded from below by $\frac{1}{3} \cdot \frac{1}{T_{init}+n+1}$, and the expected time to do such a step is at most $3 \cdot (T_{init}+n+1)$. Since T_{init} is the number of leaves in the initial tree, there are T_{init} repetitions required to delete all the variables. This implies that the expected time for the empty tree to be included in the population is upper bounded by

$$T_{init} \cdot 3 \cdot (T_{init} + n + 1) = O\left(T_{init}^2 + nT_{init}\right) \quad \square$$

THEOREM 8. *Starting with a single arbitrary initial solution of size T_{init}, the expected optimization time of SMO-GP-single on MO-WORDER and MO-WMAJORITY is $O(T_{init}^2 + n^2 T_{init} + n^3)$.*

PROOF. In the following steps, we will bound the time needed to discover the whole Pareto front, once the empty solution is introduced into the population. As shown in Theorem 1, the empty tree is included in the population after $O(T_{init}^2 + nT_{init})$ steps. The empty tree is now a Pareto optimal solution with complexity $C(X) = 0$. Note that a solution of complexity j, $0 \leq j \leq n$ is Pareto optimal in MO-WORDER and MO-MAJORITY, if it contains only the largest j variables.

[1]as the empty tree is Pareto optimal with $F(X) = C(X) = 0$

Let us assume that the population contains all Pareto optimal solutions with complexities j, $0 \leq j \leq i$. Then, a population which includes all Pareto optimal solutions with complexities j, $0 \leq j \leq i+1$, can be achieved by producing a solution Y that is Pareto optimal and that has complexity $i+1$. Y can be obtained from a Pareto optimal solution X with $C(X) = i$ and $F(X) = \sum_{i=1}^{k} w_i$ by inserting the leaf x_{k+1} (with associated weight w_{k+1}) at any position. This operation produces from a solution of complexity i a solution of complexity $i+1$.

Based on this idea we can bound the expected optimization time once we can bound the probability for such steps to happen. Choosing X for mutation has the known probability of at least $\frac{1}{T_{init}+n+1}$ as the population size is upper bound by $T_{init} + n + 1$. Next, the inserting operation of the mutation operator is chosen with probability $\frac{1}{3}$. As a specific element out of a total of n elements has to be inserted at a randomly chosen position, the probability to do so it $\frac{1}{n}$. Thus, the total probability of such a generation is lower bounded by $\frac{1}{T_{init}+n+1} \cdot \frac{1}{3} \cdot \frac{1}{n}$.

Now, we use the method of fitness-based partitions Wegener (2002) according to the $n+1$ different Pareto front sizes i. Thus, as there are only n Pareto-optimal improvements possible once the empty solution is introduced into the population, the expected time until all Pareto optimal solutions have been generated is:

$$\begin{aligned}\sum_{i=0}^{n-1} \left(\frac{1}{T_{init}+n+1} \cdot \frac{1}{3} \cdot \frac{1}{n}\right)^{-1} &= 3n^2(T_{init} + n + 1) \\ &= O(n^2 T_{init} + n^3). \quad \square\end{aligned}$$

Summing up the number of steps to generate the empty tree and all other Pareto optimal solutions, the expected optimisation time is $O(T_{init}^2 + n^2 T_{init} + n^3)$

5.2 SMO-GP-multi

In principle, when considering WORDER and WMAJORITY in SMO-GP, it is possible to generate an exponential number of trade-offs between solution fitness and complexity. For instance, if $w_i = 2^{n-i}, 1 \leq i \leq n$, then one can easily generate 2^n different individuals by considering different subsets of variables. These solutions are dominated by the true Pareto front, and our experimental analysis suggests that many of them get discarded early in the optimization process.

In this section, we employ the P_{max} measure collected in our experimental analysis (see Figure 7) to consolidate some of the theoretical bounds about WORDER and WMAJORITY presented in Neumann (2012) and introduce new bounds for their multi-operation variants.

LEMMA 2. *The expected time before SMO-GP, initialized with T_{init} leaves, adds the empty tree to its population is bound by $O(T_{init} P_{max})$.*

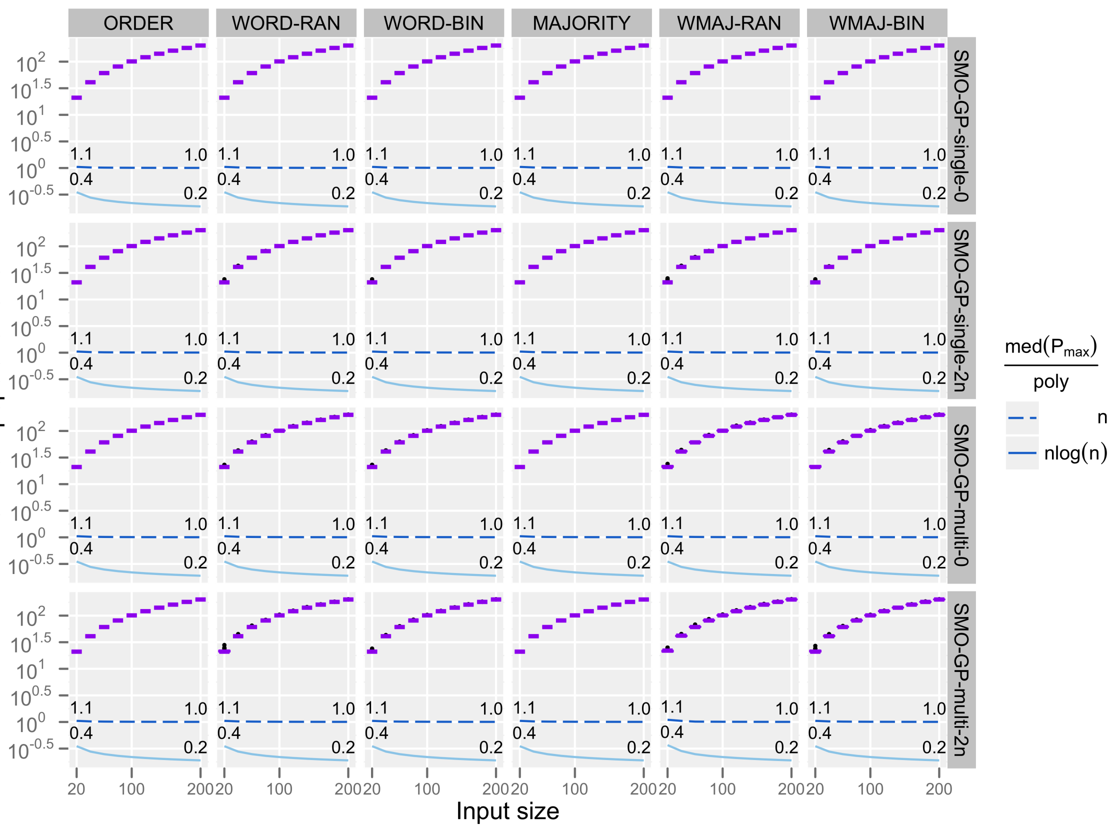

Figure 7: Maximum population size distributions (as box-plots) reached before finding an optimal solution in SMO-GP. The blue lines represent the medians divided by the corresponding polynomials and give an indication of the order of magnitude of the P_{max} measure.

PROOF. If the empty tree is the initial solution, i.e. $T_{init} = 0$, then Lemma 2 follows immediately.

If $T_{init} > 0$, then we need to perform T_{init} deletions in order to reach the empty tree. Let X be the individual of lowest complexity, then the probability of selecting it for mutation is at least $1/P_{max}$. The probability of performing a single deletion is lower bounded by $\frac{1}{3e}$ for both the single- and the multi-operation variants of the HVL-Prime operator. Therefore, the total probability of selecting the solution of lowest complexity and deleting one of its nodes is $\Omega(1/P_{max})$, and the expected time to make such a step is $O(P_{max})$. Since we need to delete T_{init} leaves, the total expected time to add the empty tree to the population is bounded by the repeated deletion taking $O(T_{init}P_{max})$ steps. $\square$

Note that, contrary to Theorem 1, this lemma holds for both the single- and the multi-mutation variants of SMO-GP. However, this lemma depends on P_{max}, which is not under the control of the user.

We now state an upper bound for SMO-GP on WORDER and WMAJORITY in both single and multi-operation variants.

THEOREM 9. *The expected optimization time of SMO-GP on WORDER and WMAJORITY is $O(P_{max}(T_{init} + n^2))$.*

PROOF. This proof follows the structure of the proof of Theorem 8. First, due to Lemma 2 we can assume that we have added the empty tree to the population after $O(T_{init}P_{max})$ steps. We now recall that a solution of complexity j, $0 \leq j \leq n$, is Pareto optimal for MO-WORDER and MO-WMAJORITY, if it contains, for the j largest weights, exactly one positive variable.

Similar to the second step of the proof of Theorem 8, we assume that the population already contains the i Pareto optimal solutions with complexities j, $0 \leq j \leq i$. Then, a population which includes all Pareto optimal solutions with complexities j, $0 \leq j \leq i+1$, can be achieved by producing a solution Y that is Pareto optimal and that has complexity $i+1$. Y can be obtained from a Pareto optimal solution X with $C(X) = i$ and $F(X) = \sum_{i=1}^{k} w_i$ by inserting the leaf x_{k+1} (with associated weight w_{k+1}) at any position. This operation produces from a solution of complexity i a solution of complexity $i+1$.

Again, based on this idea, we can bound the expected optimization time once we can bound the probability for such steps to happen. Choosing X for mutation is lower bounded by $1/P_{max}$, the inserting operation of the mutation operator is chosen with probability $1/3e$, and as a specific element out of a total of up to $2n$ elements has to be inserted at a randomly chosen position, the probability to do so it $1/2n$. Thus, the total probability of such a generation is lower bounded by $1/(6enP_{max})$.

Now, using the method of fitness-based partitions according to the $n+1$ different Pareto front sizes i, the expected time until all Pareto optimal solutions have been generated (once the empty tree was introduced) is bounded by

$$\sum_{i=0}^{n-1} \left(\frac{1}{P_{max}} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \right)^{-1} = 6en^2 P_{max}$$
$$= O(n^2 P_{max}).$$

Therefore, the total expected optimization time when starting from an individual with T_{init} leaves is thus $O(T_{init}P_{max} + n^2 P_{max})$, which concludes the proof. $\square$

As we mentioned before, to date there is no theoretical understanding about how P_{max} grows during an optimization run and, in principle, its order of growth could be exponential in n. For this reason, similarly to what we did for T_{max}, we have investigated this measure experimentally. The experimental setup is the same described previously, and the empirical distributions of P_{max} are shown as box-plots in Figure 7. The blue-toned line plots show the median P_{max} divided by n and $\log(n)$. Again, the nearly constant behavior of the solid line suggests a magnitude of P_{max} which is linearly dependent on n and, in general, very close to n as the factors, at most 1.1, suggest.

6. CONCLUSIONS

In this paper, we carried out theoretical investigations to complement the recent results on the runtime of two genetic programming algorithms (Durrett et al., 2011; Neumann, 2012; Urli et al., 2012). Crucial measures in the theoretical analyses are the maximum tree size T_{max} that is attained during the run of the algorithms, as well as the maximum population size P_{max} when dealing with multi-objective models.

We introduced several new bounds for different GP variants on the different versions of the problems WORDER and WMAJORITY. Tables 2 and 3 summarise our results and the existing known bounds on the investigated problems.

Despite the significant theoretical and experimental effort, the following open challenges still remain:

1. Almost no theoretical results for the multi-operation variants are known to date. A major reason for this is that it is not clear how to bound the number of accepted operations when employing the multi-operation variants on the weighted cases.

2. Of particular difficulty seems to be the problem to bound runtimes on WMAJORITY. There, in order to achieve a fitness increment, the incrementing operation needs to be preceded by a number of operations that reduce the difference in positive and negative occurrences of a variable. Said difference is currently difficult to control, as it can increase and decrease during a run.

3. Several bounds rely on T_{max} or P_{max} measures, which are not set in advance and whose probability to increase or decrease during the optimization is hard to predict. This is because their growth (or decrease) depends on the content of the individual and the content of the population at a specific time step. Solving these problems would allow bounds which only depend on parameters the user can initially set. Observe that limiting T_{max} and P_{max} is not an option, since we are considering variable length representation algorithms, and populations that represent *all* the trade-offs between objectives.

4. Finally, it is not known to date how tight these bounds are.

References

Doerr, B., Johannsen, D., and Winzen, C. (2010). Multiplicative drift analysis. In Pelikan, M. and Branke, J., editors, *GECCO*, pages 1449–1456. ACM.

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

Durrett, G., Neumann, F., and O'Reilly, U.-M. (2011). Computational complexity analysis of simple genetic programing on two problems modeling isolated program semantics. In *FOGA*, pages 69–80. ACM.

Goldberg, D. E. and O'Reilly, U.-M. (1998). Where does the good stuff go, and why? How contextual semantics influences program structure in simple genetic programming. In *EuroGP*, volume 1391 of *LNCS*, pages 16–36. Springer.

Kötzing, T., Sutton, A. M., Neumann, F., and O'Reilly, U.-M. (2012). The max problem revisited: the importance of mutation in genetic programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 1333–1340, New York, NY, USA. ACM.

Neumann, F. (2012). Computational complexity analysis of multi-objective genetic programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 799–806, New York, NY, USA. ACM.

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

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

Urli, T., Wagner, M., and Neumann, F. (2012). Experimental supplements to the computational complexity analysis of genetic programming for problems modelling isolated program semantics. In *PPSN*. Springer. (to be published).

Wagner, M. and Neumann, F. (2012). Parsimony pressure versus multi-objective optimization for variable length representations. In *PPSN*. Springer. (to be published).

Wegener, I. (2002). Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In *Evolutionary Optimization*, pages 349–369. Kluwer.

F(X)	(1+1)-GP, F(X)	
	k=1	k=1+Pois(1)
ORDER	$O(nT_{max})$ Durrett et al. (2011) $O(T_{init}+n\log n)$ Urli et al. (2012)	$O(nT_{max})$ Durrett et al. (2011) $O(T_{init}+n\log n)$ Urli et al. (2012)
WORDER	$O(nT_{max})\star$ $O(T_{init}+n\log n)$ Urli et al. (2012)	$O(nT_{max}(\log n+\log w_{max}))\star$ $O(T_{init}+n\log n)$ Urli et al. (2012)
MAJORITY	$O(n^2T_{max}\log n)$ Durrett et al. (2011) $O(T_{init}+n\log n)$ Urli et al. (2012)	**?** $O(T_{init}+n\log n)$ Urli et al. (2012)
WMAJORITY	**?** $O(T_{init}+n\log n)$ Urli et al. (2012)	**?** $O(T_{init}+n\log n)$ Urli et al. (2012)

F(X)	(1+1)-GP, MO-F(X)	
	k=1	k=1+Pois(1)
ORDER	$O(T_{init}+n\log n)$Neumann (2012) $O(T_{init}+n\log n)$ Urli et al. (2012)	$O(n^2\log n)\ \star\dagger$ $O(T_{init}+n\log n)$ Urli et al. (2012)
WORDER	$O(T_{init}+n\log n)$Neumann (2012) $O(T_{init}+n\log n)$ Urli et al. (2012)	**?** $O(T_{init}+n\log n)$ Urli et al. (2012)
MAJORITY	$O(T_{init}+n\log n)$Neumann (2012) $O(T_{init}+n\log n)$ Urli et al. (2012)	**?** $O(T_{init}+n\log n)$ Urli et al. (2012)
WMAJORITY	$O(T_{init}+n\log n)$Neumann (2012) $O(T_{init}+n\log n)$ Urli et al. (2012)	**?** $O(T_{init}+n\log n)$ Urli et al. (2012)

Table 2: Summary of our bounds ($\star$) and the existing theoretical upper bounds from Table 1. The average-case conjectures from Urli et al. (2012) are listed to indicate tightness. Note that (?) mark the cases for which no theoretical bounds are known, and bounds marked with ($\dagger$) require special initialisations.

F(X)	SMO-GP, MO-F(X)	
	k=1	k=1+Pois(1)
ORDER	$O(nT_{init}+n^2\log n)$Neumann (2012) $O(nT_{init}+n^2\log n)$ Urli et al. (2012)	$O(nT_{init}+n^2\log n)$Neumann (2012) $O(nT_{init}+n^2\log n)$ Urli et al. (2012)
WORDER	$O(T_{init}^2+n^2T_{init}+n^3)\star$ $O(n^3)$ Neumann (2012) $\dagger$ $O(nT_{init}+n^2\log n)$ Urli et al. (2012)	$O(T_{init}P_{max}+n^2P_{max})\star$ $O(nT_{init}+n^2\log n)$ Urli et al. (2012)
MAJORITY	$O(nT_{init}+n^2\log n)$Neumann (2012) $O(nT_{init}+n^2\log n)$ Urli et al. (2012)	$O(nT_{init}+n^2\log n)$Neumann (2012) $O(nT_{init}+n^2\log n)$ Urli et al. (2012)
WMAJORITY	$O(T_{init}^2+n^2T_{init}+n^3)\star$ $O(n^3)$ Neumann (2012) $\dagger$ $O(nT_{init}+n^2\log n)$ Urli et al. (2012)	$O(T_{init}P_{max}+n^2P_{max})\star$ $O(nT_{init}+n^2\log n)$ Urli et al. (2012)

Table 3: Summary of our bounds ($\star$) and the existing theoretical upper bounds from Table 1. The average-case conjectures from Urli et al. (2012) are listed to indicate tightness. Note that the bounds marked with ($\dagger$) require special initialisations.