Experimental Supplements to the Computational Complexity Analysis of Genetic Programming for Problems Modelling Isolated Program Semantics

Tommaso Urli[1], Markus Wagner[2], and Frank Neumann[2]

[1] DIEGM, Università degli Studi di Udine, 33100 Udine, Italy
[2] School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia

Abstract. In this paper, we carry out experimental investigations that complement recent theoretical investigations on the runtime of simple genetic programming algorithms [3, 7]. Crucial measures in these theoretical analyses are the maximum tree size that is attained during the run of the algorithms as well as the population size when dealing with multi-objective models. We study those measures in detail by experimental investigations and analyze the runtime of the different algorithms in an experimental way.

Keywords: genetic programming, problem complexity, multiple-objective optimization, experimental evaluation.

1 Introduction

In the last decade, Genetic Programming algorithms have found various applications [8] in a number of domains, however their behaviour is hard to understand in a rigorous manner. Recently, the first computational complexity results have been presented for simple genetic programming algorithms [3, 7]. The algorithms that have been considered are a stochastic hill-climber called (1+1)-GP and a population-based multi-objective programming algorithm called SMO-GP that takes into account the given problem F and the complexity C of a solution. These algorithms have been analyzed on problems with isolated program semantics taken from [5] which can be seen as the analogue of linear pseudo-Boolean functions [2] known from the computational complexity analysis of evolutionary algorithms working with fixed length binary representations.

The theoretical results provided in [3, 7] bring up several questions that remain unanswered in these papers. In particular, for different combinations of algorithms and problems no (or no exact) runtime bounds are given. In our paper, we explore the different open cases and questions in an experimental way. Similar to [1, 6], this should guide further rigorous analyses by exploring the important measures within a computational complexity analysis of the algorithms and give experimental estimates on the actual runtime of the algorithms on the different problems. Our experimental investigations, will concentrate on

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 102–112, 2012.

important measures such as the maximum tree size during the run of the single-objective algorithms analyzed in [3] and the maximum population size of the multi-objective algorithm analyzed in [7]. It can be observed from the analyses carried out in these two papers, that both measures have a different implication on the runtime of the analyzed genetic programming algorithms. Other experimental results indicate that both measures do not grow large during the run of the algorithms which would imply a fast optimization process. Furthermore, our experimental results on the actual runtime of (1+1)-GP and SMO-GP indicate an efficient optimization process.

The paper is structured as follows. In Section 2, we introduce the problems and algorithms and summarize the computational complexity results for them. (1+1)-GP is experimentally investigated in Section 3 and the behavior of SMO-GP is examined in Section 4. We finish with some concluding remarks.

2 Preliminaries

In our experimental investigations, we will treat the algorithms and problems analyzed in [3, 7]. We consider tree-based genetic programming, where a possible solution to a given problem is given by a syntax tree. The inner nodes of such a tree are labelled by symbols from a function set F, and the leaves of the tree are labelled by terminals from a set T. The problems that we examine are Weighted ORDER (WORDER) and Weighted MAJORITY (WMAJORITY). For all, the only function is the binary join operation (denoted by J), and the terminal set is a set of $2n$ variables, where $\overline{x}_i$ represents the complement of x_i. Thus, $F := \{J\}$ and $T := \{x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_n, \overline{x}_n\}$. With each variable x_i, we associate a weight $w_i \in \mathbb{R}, 1 \leq i \leq n$. Thus, the variables can differ in their contribution to the fitness of a tree. Without loss of generality, we assume that $w_1 \geq w_2 \geq \ldots \geq w_n \geq 0$.

For a given syntax tree X, its computed value S is obtained by parsing the syntax tree in-order according to the problem semantics. For WORDER, x_i is contained in S iff it is present in the tree and there is no $\overline{x}_i$ that is visited in the in-order parse before x_i. For WMAJORITY, x_i is in S iff x_i occurs in the tree at least once, and at least as often as its complement $\overline{x}_i$ (see Algorithms 1 and 2). The weight w_i of a variable x_i contributes to the fitness iff x_i is positive and contained in set S. We get the problems ORDER and MAJORITY as special cases where $w_i = 1$, $1 \leq i \leq n$, holds.

Algorithm 1. WORDER(X)

input: a syntax tree X
init : an empty leaf list l, an empty statement list S
1 Parse X in-order and insert each leaf the rear of l as it is visited;
2 Generate S by parsing l front to rear and adding a leaf to S only if its complement is not yet in S;
3 WORDER $(X) = \sum_{x_i \in S} w_i$;

Algorithm 2. WMAJORITY(X)

input: a syntax tree X
init : an empty leaf list l, an empty statement list S
1 Parse X in-order and insert each leaf the the rear of l as is is visited;
2 For $1 \leq i \leq n$: if count$(x_i \in l) \geq$ count$(\overline{x}_i \in l)$ and count$(x_i \in l) \geq 1$, then add x_i to S;
3 WMAJORITY $(X) = \sum_{x_i \in S} w_i$;

As GP mechanisms, we investigate the single-objective (1+1)-GP and the Simple Multi-Objective Genetic Programming (SMO-GP) algorithm. For the (1+1)-GP, we consider the problem of computing a solution X which maximizes a given function $F(X)$. In the case of the parsimony approach, we additionally take into account the complexity $C(X)$ of a solution (measured as the total number of nodes in the tree). Here, we optimize the multi-criteria fitness function MO-F$(X) = (F(X), C(X))$ with respect to the lexicographic order, that is, MO-F$(X) \geq$ MO-F(Y) holds iff $F(X) > F(Y) \vee (F(X) = F(Y) \wedge C(X) \leq C(Y)$.

For SMO-GP, we will treat the two objective $F(X)$ and $C(X)$ as equally important and use standard notations from the field of multi-objective optimization. A solution Y *weakly dominates* a solution X (denoted by $Y \succeq X$) iff $(F(Y) \geq F(X) \wedge C(Y) \leq C(X))$. A solution Y *dominates* a solution X (denoted by $Y \succ X$) iff $((Y \succeq X) \wedge (F(Y) > F(X) \vee C(Y) < C(X))$. 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 is to compute for each objective vector of the Pareto front a Pareto optimal solution. SMO-GP starts with a single solution and keeps at any time during the optimization a set of non-dominated solutions among the set of all solutions seen so far.

Note that this trade-off between solution complexity and solution quality has successfully applied in industry tools such as Datamodeller [4].

(1+1)-GP and SMO-GP only use the mutation operator HVL-Prime to generate offspring. HVL-Prime allows for the production of trees of varying complexity, and is based on the operations *substitution*, *deletion*, and *insertion*. For an application of HVL-Prime, a parameter k has to be chosen. k determines the number of operations that HVL-Prime performs: (1) in the *single-operation* case $k = 1$ holds, (2) in the *multi-operation* case $k = 1 + \text{Pois}(1)$ holds, where Pois(1) denotes the Poisson distribution with parameter 1. We refer the reader to [3, 7] for a detailed description on HVL-Prime. Depending on the number of operations used in the mutation operator, we get the algorithms (1+1)-GP-single and SMO-GP-single and their corresponding multi-mutation variants (1+1)-GP-multi and SMO-GP-multi.

The complete algorithms are outlined in Algorithms 3 and 4.

Algorithm 3. (1+1)-GP

```
1 Choose an initial solution X;
2 repeat
3     Set Y := X;
4     Apply mutation to Y;
5     If selection favors Y over X then
      set X := Y;
```

Algorithm 4. SMO-GP

```
1 Choose an initial solution X;
2 Set P := {X};
3 repeat
4     Randomly choose X ∈ P;
5     Set Y := X;
6     Apply mutation to Y;
7     If {Z ∈ P|Z ≻ Y} = ∅ set
      P := (P \ {Z ∈ P|Z ⪰ Y}) ∪ {Y};
```

2.1 Theoretical Results

The computational complexity analysis of genetic programming analyzes the expected number of fitness evaluations until an 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 number of fitness evaluations until the whole Pareto front has been computed is analyzed and referred to as the expected optimization time. The bounds from [3, 7] are listed in Table 1. As it can be seen, all results take into account tree sizes of some kind: either the maximum tree size T_{max} during the search plays a role in the bound, or the size of the initial tree T_{init} does. It is also unknown how tight the given bounds are. The maximum tree size for (1+1)-GP and the population size for SMO-GP play a relevant role in the theoretical analysis and will be further investigated in the rest of the paper. 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 [7].

Table 1. Computational complexity results from [3, 7]

F(X)	(1+1)-GP, F(X) [3]		(1+1)-GP, MO-F(X) [7]		SMO-GP, MO-F(X) [7]	
	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)
ORD	$O(nT_{max})$	$O(nT_{max})$	$O(T_{init} + n \log n)$	?	$O(nT_{init} + n^2 \log n)$	
WORD	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?
MAJ	$O(n^2 T_{max} \log n)$	?	$O(T_{init} + n \log n)$		$O(nT_{init} + n^2 \log n)$	
WMAJ	?	?	$O(T_{init} + n \log n)$		$O(n^3)\star$	?

2.2 Experimental Setup

In the remainder of this paper, we will empirically confirm and verify the theoretical results from [3, 7]. We consider (1+1)-GP and SMO-GP, each in their single and multi-operation variants, and investigate problems of sizes n =20, 40, 60, ..., 200 (although, for space reasons, the results in tables are shown only for $n = 100$). For the initialization, we consider the schemes init_0 (empty tree) and init_{2n} (in which a $2n$ leaves tree is generated by applying $2n$ *insertion* mutations at random positions). In total, our experiments span twelve problems: WORDER and WMAJORITY in their F(X) and MO-F(X) variants. The weight settings are set as follows: (1) $w_i = 1$, $1 \leq i \leq n$, for ORDER and MAJORITY, (2) $w_i \in [0, 1]$ chosen uniformly at random, $1 \leq i \leq n$, for WORDER-RAN and WMAJORITY-RAN, and (3) $w_i = 2^{n-i}$, $1 \leq i \leq n$, for WORDER-BIN and WMAJORITY-BIN.

The following 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 has been repeated 400 times, which results in a standard error of the mean (the standard deviation of the sampling distribution) of $1/\sqrt{400} = 5\%$.

3 (1+1)-GP

3.1 Tree size

The theoretical bounds for (1+1)-GP on ORDER and MAJORITY presented in [3] depend on the maximum tree size that is encountered during the run of the algorithms. We investigate the maximum tree size experimentally in order to see whether bloat occurs when applying the algorithms. For (1+1)-GP-single using the parsimony approach, i. e. using the function MO-F(X), the difference between the solution value S and the number of leaves not preceded by their complements can not increase during the run of the algorithm [7].

First, we investigate the tree sizes typically observed during the optimization for the different (1+1)-GP algorithms. Table 2 reports results for $n = 100$, but similar results hold for the other input sizes. The maximum tree size observed during the run of (1+1)-GP on MO-F(X) when using single-operation and empty initialization is $2n-1$, which is the minimum possible size of an optimal solution. This was expected, since the algorithm can only increase the tree by a single leaf in every accepting step. These values increase by about 10-20% in the case of init_0, when multiple HVL-Prime applications are allowed per mutation step. When the acceptance criteria is weakened by switching to the F(X) variant (i.e. the current tree can be replaced by larger ones of identical fitness), then the tree sizes are about 2.5 times larger in the single-operation case, and about 3 times larger in the multi-operation case.

Similarly, when running (1+1)-GP on MO-F(X), if the population is initialized with trees of $2n$ leaves, the largest trees encountered are of size $2 \cdot (2n) - 1$, i.e. the tree size of the initial solution, in the single-operation case, and are just minimally larger (about 1%) in the multi-operation case.

Table 2. Maximum tree sizes encountered until the individual X_{max} with maximum fitness is found. Shown are median m and median interquartile ranges iqr. Here $k = 1$ and $k = 1 + Pois(1)$ refer respectively to the single and multi-operation variants.

k	F(X)	n	(1+1)-GP, F(X)				(1+1)-GP, MO-F(X)			
			init_0		init_{2n}		init_0		init_{2n}	
			m	iqr	m	iqr	m	iqr	m	iqr
k=1	ORDER	100	519	94.5	593	100	199	0	399	2
	WORDER-RAN	100	513	85	594	90	199	0	399	0.5
	WORDER-BIN	100	513	94	591	88.5	199	0	399	0
	MAJORITY	100	507	78.5	563	72	199	0	399	0
	WMAJORITY-RAN	100	499	76.5	567	74.5	199	0	399	0
	WMAJORITY-BIN	100	499	74.5	567	75	199	0	399	0
k=1+Pois(1)	ORDER	100	670	138	742	143	223	12	399	6
	WORDER-RAN	100	667	136.5	713	131	229	12	399	6
	WORDER-BIN	100	665	150.5	735	132.5	231	12	399	4
	MAJORITY	100	624	96	668	102	239	14	401	8
	WMAJORITY-RAN	100	617	104	678	116.5	241	16	401	8
	WMAJORITY-BIN	100	635	114.5	671	116.5	243	14	401	8

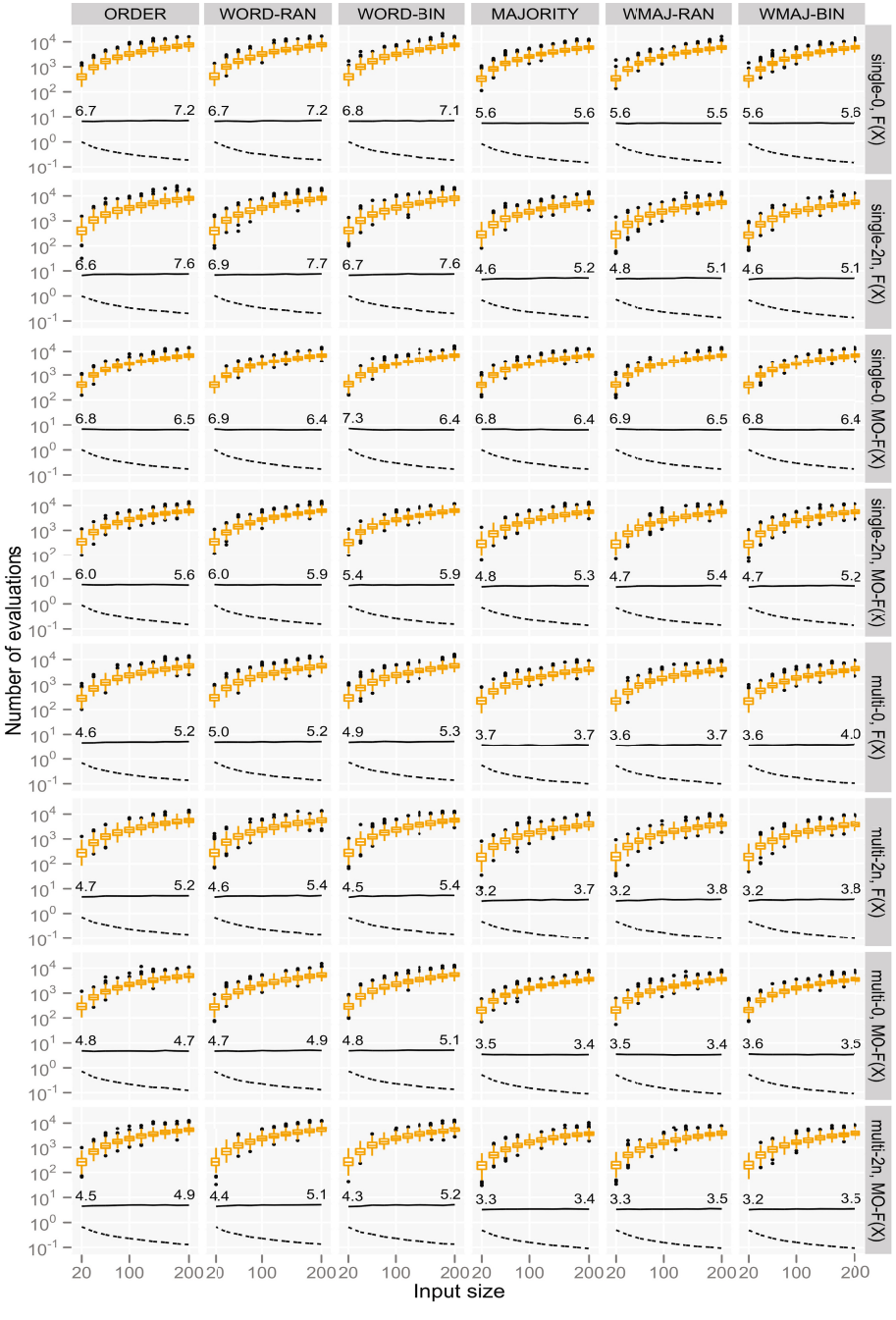

Fig. 1. Number of evaluations required by (1+1)-GP until the individual X_{max} with maximum fitness is found, shown as box plots. The solid line is the median of the number of evaluations divided by $n \log n$, the dashed line is the same median divided by n^2.

For the non-parsimony variants, however, the largest trees are about 50% larger when solving ORDER, and almost 100% when solving MAJORITY.

3.2 Runtime

Figure 1 shows the distributions of the required evaluations for the (1+1)-GP variants as box plots. The line plots represent the medians divided by different polynomials and suggest the asymptotic behavior of the algorithms: the solid line is the median number of evaluations needed to produce the individual with the optimal fitness value divided by $n \log n$, and the dashed line is the same number, but divided by n^2.

For all combinations of algorithms and problems these plots indicate an expected optimization time of $O(n \log n)$, as the solid lines closely resemble constant functions (see the y-values for $n = 20$ and $n = 200$), and the y-values of the dashed lines are decreasing with increasing values of n. The constant factor obtained by dividing the median number of evaluations by $n \log n$ is overall higher in the single-operation variants of the algorithm, suggesting that applying multi mutations can help getting earlier to the optimal solution.

One important observation is that the algorithms' asymptotic behavior appears to be same, when initialized with the empty tree, and with trees with $2n$ leaves. For the setups where a theoretical bound in Table 1 is missing, the experimental results give a strong indication about the expected optimization time being $O(n \log n)$.

4 SMO-GP

4.1 Tree Size and Population Size

Table 3 shows the maximum tree sizes and maximum population sizes that were observed up to the following two events. Firstly, until the individual X_{max} with maximum fitness is found, and secondly, until the population represents the entire true Pareto front P_{Pareto}.

It can be seen that tree and population sizes observed by SMO-GP-single are independent of the initialization. In all cases, no trees with more than the size of the Pareto optimal solution X with $F(X) = n$ (size $2n - 1$) (when using init_0) and the initial tree size $2 \cdot (2n) - 1$ (when using init_{2n}) ever belong to the population. In the multi-operation cases, the maximum population sizes are rarely higher, and the same holds for the maximum tree sizes.

4.2 Runtime

Just as in the previous section, we show now the distributions of the required evaluations as box plots in Figure 2. As before, yellow box plots represent the number of evaluations to get to X_{max}, while red box plots represent now the number of evaluations to get to P_{Pareto}. In this plot, the lines are the medians

Table 3. Maximum tree sizes and maximum population sizes encountered for SMO-GP on the MO-F(X) problem variants: (1) until the individual X_{max} with maximum fitness is found, (2) until the population represents the entire true Pareto front P_{Pareto}. Shown are median m and interquartile ranges iqr. init_0 denotes the initialization with the empty tree, and init_{2n} the one with randomly constructed trees with $2n$ leaf nodes.

		F(X)	n	maximum tree size				max. population size			
				to X_{max}		to P_{Pareto}		to X_{max}		to P_{Pareto}	
				m	iqr	m	iqr	m	iqr	m	iqr
SMO-GP, with k=1	init_0	ORDER	100	199	0	199	0	101	0	101	0
		WORDER-RAN	100	199	0	199	0	101	0	101	0
		WORDER-BIN	100	199	0	199	0	101	0	101	0
		MAJORITY	100	199	0	199	0	101	0	101	0
		WMAJORITY-RAN	100	199	0	199	0	101	0	101	0
		WMAJORITY-BIN	100	199	0	199	0	101	0	101	0
	init_{2n}	ORDER	100	399	0	399	0	101	0	101	0
		WORDER-RAN	100	399	0	399	0	101	0	101	0
		WORDER-BIN	100	399	0	399	0	101	0	101	0
		MAJORITY	100	399	0	399	0	101	0	101	0
		WMAJORITY-RAN	100	399	0	399	0	101	0	101	0
		WMAJORITY-BIN	100	399	0	399	0	101	0	101	0
SMO-GP, with k=1+Pois(1)	init_0	ORDER	100	207	6.5	207	6.5	101	0	101	0
		WORDER-RAN	100	211	8	211	8	102	2	102	1
		WORDER-BIN	100	211	6	211	6	102	1	102	2
		MAJORITY	100	215	10.5	215	10.5	101	0	101	0
		WMAJORITY-RAN	100	223	12	223	12	103	2	103	1
		WMAJORITY-BIN	100	219	10	219	10	102	2	103	2
	init_{2n}	ORDER	100	399	4	399	4	101	0	101	0
		WORDER-RAN	100	399	4	399	4	102	2	102	1
		WORDER-BIN	100	399	4	399	4	102	1	102	2
		MAJORITY	100	399	4	399	4	101	0	101	0
		WMAJORITY-RAN	100	400	6	400	6	103	2	104	2
		WMAJORITY-BIN	100	401	6	401	6	102	2	103	1

divided by different polynomials and suggest the asymptotic behavior of the algorithms: the solid line is the median number of evaluations needed to get to the Pareto front divided by $n^2 \log n$, and the dashed line is the same number, but divided by n^3. For all combinations of algorithms and the problems, these plots indicate an expected optimization time of $O(n^2 \log n)$ for ORDER and MAJORITY, as the solid lines closely resemble constant functions, and the y-values of the dashed lines are decreasing with increasing values of n. For the weighted variants, however, the solid lines appear to be slowly rising, indicating a runtime in $\Omega(n^2 \log n) \cap O(n^3)$, although the runtime is extremely close to $O(n^2 \log n)$.

Furthermore, it can be observed that there is a significant time difference, for SMO-GP-multi, between finding the individual with the optimal fitness value and finding the entire Pareto front. For SMO-GP-single, this time difference is negligible, which is the reason why the corresponding orange box plots are scarcely identifiable behind the red ones.

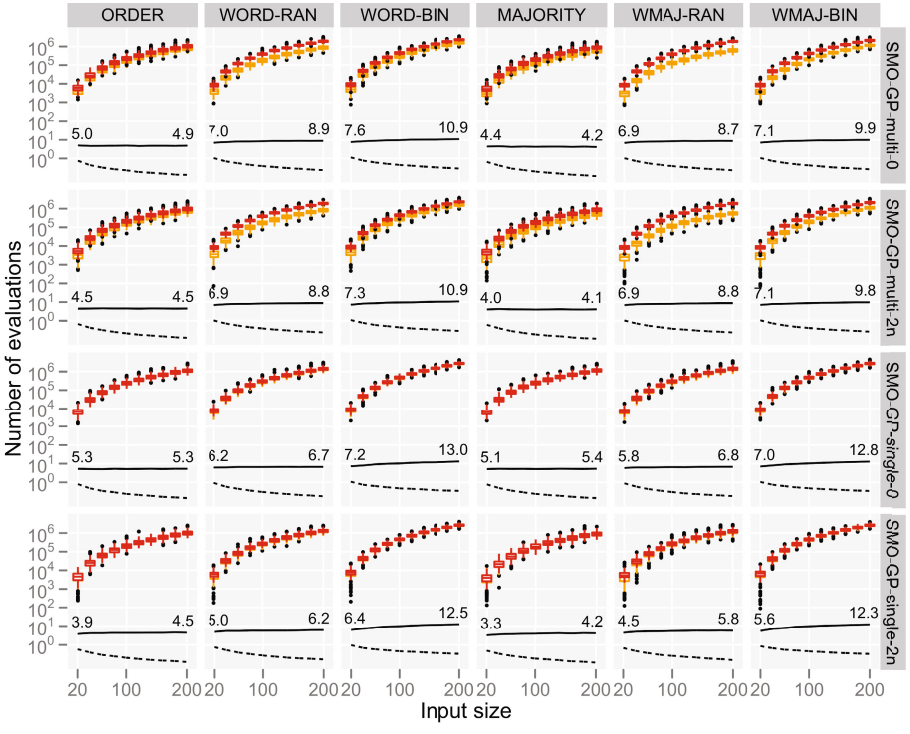

Fig. 2. Shown as box plots is the number of evaluations required: (1) until the individual X_{max} with maximum fitness is found (orange), (2) until the population represents the entire true Pareto front P_{Pareto} (red). The solid line is the median of the latter number of evaluations divided by $n^2 \log n$, the dashed line is it divided by n^3.

5 Conclusions

In this paper, we carried out experimental investigations to complement recent theoretical results on the runtime of two genetic programming algorithms [3, 7]. Crucial measures in these theoretical analyses are the maximum tree size that is attained during the run of the algorithms, as well as the population size when dealing with multi-objective models. Furthermore, virtually no theoretical results for the multi-operation variants are known to date. It is also unknown how tight the given bounds are. The analysis of our empirical investigations allowed us to fill in the gaps in the theory with conjectures about the expected optimization time (see Tables 4 and 5) of these algorithms.

Our experimental evaluation shows that the expected optimization time of (1+1)-GP F(X) is very close to $O(n \log n)$. Our results, however, are based on an initial tree size, i.e. T_{init}, which is always linear in n, and thus the T_{init} term suggested by theoretical results is always dominated by the $O(n \log n)$ term. Nevertheless, it is easy to show that by using arbirarily large initial tree sizes it is possible to obtain expected optimization times in which the T_{init} term is

Table 4. Summary of our conjectures (†) and the existing upper bounds from Table 1

F(X)	(1+1)-GP, F(X)		(1+1)-GP, MO-F(X)	
	k=1	k=1+Pois(1)	k=1	k=1+Pois(1)
ORDER	$O(nT_{max})$ [3] $O(T_{init} + n \log n)$ †	$O(nT_{max})$ [3] $O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †
WORDER	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †
MAJORITY	$O(n^2 T_{max} \log n)$ [3] $O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †
WMAJORITY	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$ †	$O(T_{init} + n \log n)$[7]	$O(T_{init} + n \log n)$ †

Table 5. Summary of our conjectures (†) and the existing upper bounds from Table 1

F(X)	SMO-GP, MO-F(X)	
	k=1	k=1+Pois(1)
ORDER	$O(nT_{init} + n^2 \log n)$[7]	$O(nT_{init} + n^2 \log n)$[7]
WORDER	$O(n^3)\star$ [7] $O(nT_{init} + n^2 \log n)$ †	$O(nT_{init} + n^2 \log n)$ †
MAJORITY	$O(nT_{init} + n^2 \log n)$[7]	$O(nT_{init} + n^2 \log n)$[7]
WMAJORITY	$O(n^3)\star$ [7] $O(nT_{init} + n^2 \log n)$ †	$O(nT_{init} + n^2 \log n)$ †

relevant. For this reason we conjecture an expected optimization time of $O(T_{init} + n \log n)$. Following the same reasoning for SMO-GP, we conjecture a runtime of $O\left(nT_{init} + n^2 \log n\right)$ by noting that the observed runtimes are very close to $O(n^2 \log n)$ and that the algorithm has to evolve a population of size $O(n)$.

As a further development for this line of research, it would be interesting to prove these conjectured bounds theoretically and to show how they are related to maximum population size reached during an optimization run.

References

[1] Briest, P., Brockhoff, D., Degener, B., Englert, M., Gunia, C., Heering, O., Jansen, T., Leifhelm, M., Plociennik, K., Röglin, H., Schweer, A., Sudholt, D., Tannenbaum, S., Wegener, I.: Experimental Supplements to the Theoretical Analysis of EAs on Problems from Combinatorial Optimization. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN VIII. LNCS, vol. 3242, pp. 21–30. Springer, Heidelberg (2004)

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

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

[4] Evolved Analytics LLC. DataModeler 8.0. Evolved Analytics LLC (2010)
[5] Goldberg, D.E., O'Reilly, U.-M.: Where Does the Good Stuff Go, and Why? How Contextual Semantics Influences Program Structure in Simple Genetic Programming. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 16–36. Springer, Heidelberg (1998)
[6] Lässig, J., Sudholt, D.: Experimental Supplements to the Theoretical Analysis of Migration in the Island Model. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 224–233. Springer, Heidelberg (2010)
[7] Neumann, F.: Computational complexity analysis of multi-objective genetic programming. In: GECCO. ACM (to be published, 2012); arxiv.org: CoRR abs/1203.4881
[8] Poli, R., Langdon, W.B., McPhee, N.F.: A Field Guide to Genetic Programming. lulu.com (2008)