Parsimony Pressure versus Multi-objective Optimization for Variable Length Representations

Markus Wagner and Frank Neumann

School of Computer Science, University of Adelaide, Adelaide, SA 5005, Australia

Abstract. We contribute to the theoretical understanding of variable length evolutionary algorithms. Such algorithms are very flexible but can encounter the bloat problem which means solutions grow during the optimization run without providing additional benefit. We explore two common mechanisms for dealing with this problem from a theoretical point of view and point out the differences of a parsimony and a multi-objective approach in a rigorous way. As an example to point out the differences, we consider different measures of sortedness for the classical sorting problem which has already been studied in the computational complexity analysis of evolutionary algorithms with fixed length representations.

1 Introduction

Evolutionary algorithms that work with a variable length representation often encounter the bloat problem which means that individuals grow without providing additional benefit to the quality of the solutions. Even worse such a growth of the individuals can block the optimization process such that problems that are relatively easy to optimize can not be handled by variable length evolutionary algorithms. Due to this problem, different methods have been introduced to deal with the bloat problem. Our goal is to study the behavior of variable length evolutionary algorithms from a mathematical perspective. We will examine algorithms for distinguished classes of problems and point out the impact of different approaches for dealing with the bloat problem in a rigorous way.

The most prominent example of a variable length evolutionary algorithm is genetic programming [7] which often evolves tree structures for a given problem. Recently, the first computational complexity results on these type of algorithm have been obtained. They follow the line of research that has successfully followed for evolutionary algorithms with fixed length representation (see the books [1, 11] for an overview). Variable length representations increase the search space significantly and in the light of genetic programming it seems to be wishful to better understand the behavior of algorithms using such representations from a theoretical point of view.

The computational complexity analysis of variable length evolutionary algorithms has started just recently. For example, Cathabard et al. [2] investigated non-uniform mutation rates for problems with unknown solution lengths. They used a simple evolutionary algorithm to find a bitstring with an unknown number of leading ones, and although the bitstring had some predetermined maximum length, only an unknown number of initial bits was used by the fitness function. Durrett et al. [3] investigated worst-case and average-case runtimes of a simple tree-based genetic programming algorithm. The tackled problems were separable, with independent and additive fitness structures. Kötzing et al. [6] analysed simple GP algorithms for the MAX problem.

One prominent way of dealing with the bloat problem is the parsimony approach. In the case, that two solutions have equal quality the solution of lower complexity is preferred. Another way of coping with the bloat problem is to use a multi-objective approach which uses in each iteration of a variable length evolutionary algorithm a population which represents the different trade-offs according to the original goal function and the complexity of a solution. The solutions that represent the trade-offs are called Pareto optimal. Note that the parsimony approach is a scalarization approach as it uses these Pareto optimality and a lexicographic ordering. It is known that each global solution is also Pareto optimal, but not all Pareto optimal solutions can necessarily be found through scalarizations (e.g., see [13]). Both approaches of coping with the bloat problem have recently been examined for the problems ORDER and MAJORITY in the context of genetic programming [9, 14].

We further explore the use of parsimony pressure and multi-objective models. In [9] it is shown that both approaches help for ORDER and MAJORITY, but the differences between these two approaches are not examined. In this paper, we point out that switching from the parsimony approach to the multi-objective one can significantly reduce the runtime. In particular, we show that the parsimony approach can have local optima which lead to an infinite runtime whereas the multi-objective approach is able to compute the optimal solution within a polynomial number of steps.

We show these results for a classical problem from the computational complexity analysis of evolutionary algorithms with fixed-length representations, namely the sorting problem (sorting). Scharnow, Tinnefeld, and Wegener [12] considered sorting as an optimization problem and investigated different fitness functions measuring the sortedness of a permutation of elements. Different fitness functions lead to problems of different difficulties. Our goal is to explore how variable-length evolutionary algorithms behave on these problems. We take it as a prominent example to discuss the differences between a parsimony approach and a multi-objective one. In particular, we show that the parsimony approach can end up for a lot of the different sortedness measures in local optima when using a variable length representation whereas the multi-objective approach allows to compute the whole Pareto front in expected polynomial time.

Our paper is organized as follows. In Section 2, we introduce the two models and the different measures of sortedness. We examine the parsimony approach in Section 3 and show that it leads to local optima in the search space. In Section 4, we show that the multi-objective approach is able to compute the whole

Mutate Y by applying k operations. For each operation, randomly choose to either substitute, insert, or delete.

– If substitute, replace a randomly chosen element of Y with a new element $u \in E$ selected uniformly at random.
– If insert, choose an element v in Y uniformly at random and select $u \in E$ uniformly at random. Randomly decide whether u is inserted before or after v in Y.
– If delete, randomly choose an element v of Y and delete it.

Fig. 1. Mutation operator

Pareto front of the underlying optimization problem in expected polynomial time. Finally, we finish with some conclusions.

2 Preliminaries

Our goal is to study the difference between a parsimony and a multi-objective approach for variable length evolutionary algorithms. Solutions contain (possibly multiple) elements from a set E of elements. We will consider mutation-based algorithms which produce new solutions by applying the mutation operator outlined in Figure 1. The mutation operator is parametrized by a parameter k which determines the number of operations applied to the individual Y. For single operations $k = 1$ holds. In the case of multiple operations, k is chosen according to $1 + Pois(1)$ where $Pois(1)$ denotes the Poisson distribution with expectation 1.

We will consider a given problem F and the complexity of a solution C measured by the number of elements in the solution. C should be minimized and we assume that F should be maximized. The notions can be easily adjusted to the minimization of a problem F, which will later on be considered.

In the parsimony approach, 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

$$F(X) \geq F(Y) \vee (F(X) = F(Y) \wedge C(X) \leq C(Y)) . \qquad (1)$$

In the multi-objective case, we treat the two criteria F and C as equally important and consider the classical Pareto dominance relations:

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 solution 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 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.

2.1 Sortedness Measures

We will analyze our algorithms on different measures of sortedness for the classical sorting problem. It can be stated as follows. Given a totally ordered set $E = \{1, \ldots, n\}$ of n elements, the task is to find a permutation π_{opt} of the elements of E such that

$$\pi_{opt}(1) < \pi_{opt}(2) < \ldots < \pi_{opt}(n)$$

holds, where $<$ is the order on E. Without loss of generality, we assume $\pi_{opt} = id$, i. e. $\pi_{opt}(i) = i$ for all i, throughout this paper.

The set of all permutations π forms a search space that has already been investigated in [12] for the analysis of permutation-based evolutionary algorithms. The authors of that paper investigate sorting as an optimization problem whose goal is to maximize the sortedness of a given permutation. We will consider the following fitness functions measuring the sortedness of a given permutation introduced in [12]:

- HAM(π), measuring the number of elements at correct position, which is the number of indices i such that $\pi(i) = i$,
- RUN(π), measuring the number of maximally sorted blocks, which is the number of indices i such that $\pi(i+1) < \pi(i)$ plus one,
- EXC(π), measuring the minimal number of pairwise exchanges in π, in order to sort the sequence.

Note that $EXC(\pi)$ can be computed in linear time, based on the cycle structure of permutations. If the sequence is sorted, it has n cycles. Otherwise, it is always possible to increase the number of cycles by exchanging an element that is not sitting at its correct position with the element that is currently sitting there. For any given permutation π consisting of $n - k$ cycles, $EXC(\pi) = k$.

We do not consider the functions INV (pairs in order) and LAS (longest ascending sequence) given in [12] as they are easy to be optimized for all the algorithms that we consider.

We will investigate the different measures for variable-length evolutionary algorithms. Consequently, we might have to deal with incomplete permutations as not all elements have to be contained in a given individual. Most measures can also be used for incomplete permutation, but we have to make sure that complete permutations always obtain a better fitness than incomplete ones. Furthermore, the sortedness measure should guide the algorithm from incomplete permutations to complete ones.

Algorithm 1. Derivation of $F(X)$

```
1 Generate $\pi$ by parsing $X$ front to rear and adding an element to $\pi$ only if it is
  not yet in $\pi$;
2 Return $F(\pi)$;
```

Algorithm 2. (1+1) GP-single for maximization

```
1 Choose an initial solution $X$;
2 repeat
3     Set $Y := X$;
4     Apply the mutation operator (given in Figure 1) with $k = 1$ to Y;
5     if $f(Y) \geq f(X)$ then set $X := Y$;
```

We will use the sortedness measures as above and use the following special fitness assignments that enforce the previously stated properties.

- RUN(π) $= n + 1$ if $|\pi| = 0$, otherwise RUN$(\pi) = b + m$ is the sum of the number of maximally sorted blocks b, and the number of elements missing $m = n - |\pi|$,
- If $|\pi| \leq n$ then EXC$(\pi) = e + m + 1$, otherwise EXC$(\pi) = e$, where e is the number of necessary exchanges e, and $m = n - |\pi|$ the number of elements missing.

Note that e can be computed for incomplete permutations as well, as only the order $<$ on E has to be respected. This means that, the permutations $\pi_1 = (1, 4)$ and $\pi_2 = (1, 2, 3, 4)$ require no changes, but EXC$(\pi_1) \neq$ EXC(π_2), as the number of missing elements differs.

For example, for a tree X with $\pi = (2, 3, 4, 5, 1, 6)$ and $n = 7$, the sortedness results are HAM$(X) = 1$, RUN$(X) = 2 + 1 = 3$, and EXC$(X) = 4 + 1 + 1 = 6$.

We now define our multi-objective variants of sorting. When adding the complexity of a data-structure as the second measure, we get the problems MO-HAM, MO-RUN, and MO-EXC, respectively. Given a variable length solution X and a problem F, we will refer by $F(X) = F(\pi)$ to its fitness. Here π is obtained from X (see Algorithm 1) by parsing X and adding an element x to π if it is not yet contained in it.

3 Local Optima and the Parsimony Approach

In this section, we consider simple variable length evolutionary algorithms using the parsimony approach. To stress the use of variable-length representations and to make the connection to recent investigations on the computational complexity of genetic programming [3, 9], we will view our algorithms as simple genetic programming algorithms.

The single-objective variant called (1+1) GP-single starts with an initial solution X, and produces in each iteration one single offspring Y by applying the mutation operator given in Figure 1 with $k = 1$. This means that it is a stochastic hillclimber which explores its local neighborhood. In the case of maximization, Y replaces X if $f(Y) \geq f(X)$ holds. Minimization problems are tackled in the analogous way.

In the following, we show that the parsimony approach leads to local optima for various types of sortedness measure.

Let $I_1 = (n, 2, 3, \ldots, n-3, n-2, n-1, 1)$ be the initial solution. We point out that this is a local optimum for (1+1) GP-single on MO-EXC leading to an infinite optimization time.

Theorem 1. *Let I_1 be an initial solution. Then the optimization time of (1+1) GP-single on MO-EXC is infinite.*

Proof. The individual I_1 has an EXC-value of 1 and a length of n. In order to reduce the fitness down to 0, it would be necessary to move the n from the head of the permutation to its end.

For this to happen, deletions and substitutions cannot be considered, as they would produce incomplete permutations, and incomplete permutations have EXC-values of at least 2.

Similarly, this situation cannot be solved using a single insertion: it is not possible to introduce n at its correct position within the permutation, as the existing n is preventing the new one from becoming expressed.

Therefore, it it not possible to improve the number of elements sitting at their correct (relative) position via a single mutation. Thus, (1+1) GP-single takes infinitely long, when initialized with I_1. □

We continue by investigating the sortedness measure RUN. Without loss of generality, let n be even and $I_2 = \left(\frac{n}{2}+1, \frac{n}{2}+2, \ldots, n-1, n, 1, 2, \ldots, \frac{n}{2}-1, \frac{n}{2}\right)$ be an initial solution. The following theorem shows that I_2 is a local optimum for (1+1) GP-single on MO-RUN.

Theorem 2. *Let I_2 be the initial solution. Then the optimization time of (1+1) GP-single on MO-RUN is infinite.*

Proof. The individual I_2 has a RUN-value of 2, which cannot be improved via a single insertion: $n/2$ elements have to change their positions in the inorder parsed list that is used for the computation of the RUN-value. Furthermore, a single deletion or substitution results in a worse sortedness value as one element is then missing (as defined in Section 2.1). Therefore, the runtime of the single-operation case of (1+1) GP is infinite, when initialised with this particular individual. □

Finally, we consider the sortedness measure *HAM* and investigate the initial solution $I_3 = (1, n-2, 3, 4, 5, \ldots, n-3, 2, n-1, n)$. We show that this is a local optimum for MO-HAM.

Theorem 3. *Let I_3 be the initial solution. Then the optimization time of (1+1) GP-single on MO-HAM is infinite.*

Algorithm 3. SMO-GP

```
1 Choose an initial solution $X$;
2 Set $P := \{X\}$;
3 repeat
4     Choose $X \in P$ uniformly at random;
5     Set $Y := X$;
6     Apply mutation to Y;
7     if $\{Z \in P \mid Z \succeq Y\} = \emptyset$ then set $P := (P \setminus \{Z \in P \mid Z \succ Y\}) \cup \{Y\}$;
```

Proof. The individual I_3 has the elements 2 and $n-2$ at incorrect positions, resulting in a HAM-value of $n-2$. It is not possible to maintain the HAM-value (or improve it) via deletions, as they decrease the number of elements at correct positions. Substitutions can also not maintain the HAM-value. A substitution of the $n-2$ by a 2 would result in the elements to the right to shift away from their correct position as in both cases the element at the third position would no longer get expressed. This leaves only the option of using insertions, in order to generate an individual that is accepted. The element $n-2$ cannot be introduced successfully at its correct position as its current occurrence is blocking a later expression. If the 2 is introduced at its correct position, then the resulting permutation is $(1, 2, n-2, 3, 4, 5, \ldots, n-3, n-1, n)$ as the second 2 is no longer gets expressed, and the corresponding HAM-value for this permutation is 4.

Thus, the runtime of the single-operation case of (1+1) GP is infinite, when initialised with this particular individual. □

4 Multi-objective Approach

We consider the Simple Evolutionary Multi-Objective Genetic Programming (SMO-GP) algorithm introduced in [9] and motivated by the SEMO algorithm for fixed length representations of Laumanns et al. [8]. Variants of SEMO have been frequently used in the runtime analysis of evolutionary multi-objective optimization for fixed length representations [4, 5, 10, 11]. SMO-GP (see Algorithm 3) is a population-based approach that starts with a single solution and keeps in each iteration a set of non-dominated solutions obtained during the optimization run. In each iteration, it picks one solution uniformly at random and produces one offspring Y by mutation. Y is introduced into the population iff it is not weakly dominated by any other solution in P. If Y is added to the population all individuals that are dominated by Y are discarded.

SMO-GP-single uses the mutation operator given in Figure 1 with k=1. We also consider SMO-GP-multi which differs from SMO-GP-single by choosing k according to $1 + Pois(1)$.

In this section, we analyze the performance of the SMO-GP variants on each one of the fitness functions introduced in Section 2.1. In particular, we analyze the expected number of iterations to compute the optimal solution. We call this the *expected optimization time* of the algorithms.

The following lemma bounds the expected time until the empty solution has been included into the population, when considering an arbitrary optimization problem:

Lemma 1 (Neumann [9]). *Let I_{init} be the size of the initial solution and k be the number of different fitness values of a problem F. Then the expected time until the population of SMO-GP-single and SMO-GP-multi applied to MO-F contains the empty solution is $O\left(kI_{init}\right)$.*

Theorem 4. *The expected optimization time of SMO-GP-single and SMO-GP-multi is $O(nI_{init} + n^3 \log n)$ on MO-EXC and MO-RUN and $O(nI_{init} + n^4)$ on MO-HAM.*

Note that for all three problems, only solutions of complexity i, $0 \leq i \leq n$ can be Pareto optimal. For RUN, these are solutions X with $C(X) = i$ and $\text{RUN}(X) = n + 1 - i$, $0 \leq i \leq n$.

Proof. In the following, we will first prove the theorem for MO-RUN. The proofs for MO-EXC and MO-HAM follow the same structure.

RUN has $n+1$ different fitness values. Using Lemma 1, the empty solution is produced after an expected number of $O\left(kI_{init}\right)$ steps.

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. 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 $C(X) = i$ by inserting any of the $n-i$ missing elements into the correct 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 probability at least $1/(n+1)$ as the population size is upper bound by $n+1$. Next, the mutation step carrying out just one operation happens with at least $1/e$, and the inserting operation of the mutation operator is chosen with probability $1/3$. As $n-i$ out of the n elements are missing, any of those can be inserted. However, the correct position for such a randomly chosen element has to be chosen, in order to produce the Pareto optimal solution of complexity $i+1$. This probability is at least $1/2 \cdot 1/n$, as the number of leaf nodes is bound by n, and the probability to insert as the correct child of the newly introduced inner node is at least $1/2$. Thus, the total probability of such a generation is

$$\frac{1}{n+1} \cdot \frac{1}{3e} \cdot \frac{1}{2n} \cdot \frac{n-i}{n}.$$

Now, we use the method of fitness-based partitions [15] according to the $n+1$ different fitness values of 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:

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

Taking into account the expected time to produce the empty solution, the expected time until the whole Pareto front of MO-RUN has been computed is $O\left(nI_{init} + n^3\log n\right)$.

The proof for MO-EXC follows the same structure. First, note that if the ordering within the permutation requires an exchange, then this individual is dominated by individuals of same complexity that require fewer exchanges. Just as with MO-EXC, let us assume that the population contains all Pareto optimal solutions with complexities j, $0 \le j \le i$. Then, a population which includes all Pareto optimal solutions with complexities j, $0 \le j \le i+1$, can be achieved by inserting any of the $n-i$ missing elements into the correct position of the Pareto optimal individual X with $C(X) = i$. The probability for such a step to happen is at least $\frac{1}{n+2}\cdot\frac{1}{3e}\cdot\frac{1}{2n}\cdot\frac{n-i}{n}$. Now, as $n+2$ different EXC-values are possible, and by summing up the waiting times as done for MO-RUN, the expected optimization time is $O(nI_{init} + n^3\log n)$.

Similarly, we can prove an upper bound for MO-HAM. First, note that each Pareto optimal solution with HAM-value i represents a perfectly sorted permutation of the i elements $1,\ldots,i$. Just as above, let us assume that the population contains all Pareto optimal solutions with complexities j, $0 \le j \le i$. Then, a population which includes all Pareto optimal solutions with complexities j, $0 \le j \le i+1$, can be achieved by inserting the element $i+1$ into its correct position (i. e., as the rightmost leaf) in the Pareto optimal individual X with $\text{HAM}(X) = C(X) = i$. The probability for such a step to happen is at least $\frac{1}{n+1}\cdot\frac{1}{3e}\cdot\frac{1}{2n}\cdot\frac{1}{n} = \Omega\left(\frac{1}{n^3}\right)$ and the corresponding waiting time is $O(n^3)$. There are $n+1$ different HAM-values. This implies that the expected optimization time is $O(nI_{init} + n^4)$. $\square$

5 Conclusions

Variable length representations are frequently used in evolutionary algorithms. The most prominent example using such a representation is genetic programming. With this paper, we have contributed to the theoretical understanding when using such a representation. We discussed two methods for dealing with bloat which frequently occurs when using such a representation. To point out the differences between these two approaches, we examined different measures of sortedness that have been analyzed for evolutionary algorithms with fixed length representations. Our analysis for the parsimony approach shows that variable length representations might have difficulties when dealing with simple measures of sortedness due to the presence of local optima. Our runtime analysis for simple multi-objective algorithms shows that they compute the whole Pareto front for the examined sortedness measures in expected polynomial time.

References

[1] Auger, A., Doerr, B. (eds.): Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific (2011)

[2] Cathabard, S., Lehre, P.K., Yao, X.: Non-uniform mutation rates for problems with unknown solution lengths. In: FOGA, pp. 173–180. ACM, New York (2011)

[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] Friedrich, T., He, J., Hebbinghaus, N., Neumann, F., Witt, C.: Approximating covering problems by randomized search heuristics using multiobjective models. Evolutionary Computation 18(4), 617–633 (2010)

[5] Giel, O., Lehre, P.K.: On the effect of populations in evolutionary multiobjective optimisation. Evolutionary Computation 18(3), 335–356 (2010)

[6] Kötzing, T., Sutton, A., Neumann, F., O'Reilly, U.-M.: The Max problem revisited: the importance of mutation in genetic programming. In: GECCO (to appear, 2012)

[7] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)

[8] Laumanns, M., Thiele, L., Zitzler, E.: Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. IEEE Trans. Evolutionary Computation 8(2), 170–182 (2004)

[9] Neumann, F.: Computational complexity analysis of multi-objective genetic programming. In: GECCO (to appear, 2012), `http://arxiv.org/abs/1203.4881`

[10] Neumann, F., Wegener, I.: Minimum spanning trees made easier via multi-objective optimization. In: GECCO, pp. 763–770. ACM Press (2005)

[11] Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity. Springer (2010)

[12] Scharnow, J., Tinnefeld, K., Wegener, I.: The analysis of evolutionary algorithms on sorting and shortest paths problems. Journal of Mathematical Modelling and Algorithms 3, 349–366 (2004)

[13] Shukla, P.K., Deb, K.: On finding multiple Pareto-optimal solutions using classical and evolutionary generating methods. European Journal of Operational Research 181(3), 1630–1652 (2007)

[14] Urli, T., Wagner, M., Neumann, F.: Experimental Supplements to the Computational Complexity Analysis of Genetic Programming for Problems Modelling Isolated Program Semantics. In: Coello Coello, C.A., et al. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 102–112. Springer, Heidelberg (2012)

[15] Wegener, I.: Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In: Evolutionary Optimization. International Series in Operations Research and Management Science, vol. 48, pp. 349–369. Springer, US (2003)