

# Computational Complexity, Genetic Programming, and Implications

Bart Rylander, Terry Soule, and James Foster

Initiative for Bioinformatics and Evolutionary Studies (IBEST)

Department of Computer Science, University of Idaho

Moscow, Idaho 83844-1014 USA

rylander@up.edu, {soule,foster}@csuidaho.edu

**Abstract.** Recent theory work has shown that a Genetic Program (GP) used to produce programs may have output that is bounded above by the GP itself [1]. This paper presents proofs that show that 1) a program that is the output of a GP or any inductive process has complexity that can be bounded by the Kolmogorov complexity of the originating program; 2) this result does not hold if the random number generator used in the evolution is a true random source; and 3) an optimization problem being solved with a GP will have a complexity that can be bounded below by the growth rate of the minimum length problem representation used for the implementation. These results are then used to provide guidance for GP implementation.

## 1 Introduction

Informally, computational complexity is the study of classes of problems based on the rate of growth of space, time, or other fundamental unit of measure as a function of the size of the input. It seeks to determine what problems are computationally tractable, and to classify problems based on their best possible convergence time over all possible problem instances and algorithms. A complexity analysis of a method of automatic programming such as a Genetic Algorithm (GA) or a GP seeks to answer another question as well. Namely, does the method in question, with all of its method-specific computational baggage, provide an advantage or disadvantage to solving a particular problem over all other methods. This is in contrast to a study that seeks to identify what features of a problem instance cause the method to converge more slowly, or a study that seeks to analyze the expected convergence time of the method itself.

To illustrate this difference, consider the well-known problem of Sort (i.e. given a list of  $n$  elements, arrange the list in some predefined manner). If there is only one element that is out of order then a particular algorithm may converge more quickly than if all the elements are out of order, as is the case with the algorithm Insertion Sort. An analysis based on problem instances is concerned with what features of the particular instance cause the convergence to occur more slowly (such as the number of elements being out of order). An analysis of the algorithm itself may come to the

conclusion that Insertion Sort is  $\bigcup O(n^2)$ . The *computational complexity* analysis is concerned with the complexity of the *problem* over all possible instances, sizes, and algorithms. In this case, the complexity of Sort is the complexity of the fastest *possible* algorithm for solving Sort, not just the fastest *known* algorithm.

By confining this analysis to the complexity of problems specifically when an evolutionary algorithm is applied, we can compare our findings with what is already known about the complexity of problems. In this way, we may be able to ascertain when best to apply evolutionary algorithms. This paper is a continuation of an ongoing effort to understand the computational complexity of problems specific to evolutionary algorithms. In particular, it is a report that provides proofs and analysis that describe the complexity of problems as they relate to GPs and the implications of these results.

GP is a biologically inspired method for evolving programs to solve particular problems. In addition to evolving programs, GPs have been successfully used to optimize functions [2]. It is difficult to characterize the complexity of a problem specific to a method of programming. Holding all things constant, you measure what must change as the size of the input instance increases. It is even more difficult to describe the complexity of a problem that can be solved by a program that is *itself* the output of a program, as is the case with the typical GP. In general, this type of question cannot be answered. What *can* be done however, is to compare the information content of a program with the information content of its output and in this way provide a bound on the complexity of that output. This, in addition to an analysis for function optimization, is what will be described below.

## 2 Kolmogorov Complexity

Kolmogorov complexity analysis is uniquely suited for addressing this problem. It was created, among other reasons, to provide a way to evaluate objects statically. This section details some of the definitions and theorems of Kolmogorov complexity that are pertinent to this study.

Informally, the amount of information in a finite object (like a string) is the size in bits of the smallest program that, starting with a blank memory, outputs the string and then terminates. In the case of an infinite string, it is the smallest program that can continue to output the infinite series. As an example, consider pi. It is a provably infinite series. And yet, despite this infinity, it can be represented in much less than an infinite string. Another example is an infinite sequence of 1's. Though the sequence of 1's is infinite, a program to produce it can be written in far fewer bits. See Figure 1.

```
while(true){cout<<<<"1";}
```

**Fig. 1.** Sample program to output an infinite series of 1's

By this type of measure, a number such as 9,857,384,002 is more complex than the number 10,000,000,000 even though the second number is larger than the first. This is