GA-Hardness Revisited

Haipeng Guo and William H. Hsu

Department of Computing and Information Sciences
Kansas State University, Manhattan, KS 66502
{hpguo,bhsu}@cis.ksu.edu

1 Introduction

Ever since the invention of Genetic Algorithms (GAs), researchers have put a lot of efforts into understanding what makes a function or problem instance hard for GAs to optimize. Many measures have been proposed to distinguish so-called *GA-hard* from *GA-easy* problems. None of these, however, has yet achieved the goal of being a reliable predictive GA-hardness measure. In this paper, we first present a general, abstract theoretical framework of instance hardness and algorithm performance based on Kolmogorov complexity. We then list several major misconceptions of GA-hardness research in the context of this theory. Finally, we propose some future directions.

2 Instance Hardness and Algorithm Performance

Intuitively, an algorithm a performs better if it compiles more information about the input instance. An instance f is hard if it does not have much structure information for any algorithm to exploit to solve it. Since both algorithm and problem instance can be encoded as finite strings, we can build an abstract framework of problem instance hardness and algorithm performance base on Kolmogorov complexity as follows to capture these intuitions. We measure the hardness of f by $K(f)$, the Kolmogorov complexity of f, which is defined as the size of the smallest program that can produce f. It can be seen as the absolute information, or the "randomness", of f. The information in a about f is defined by $I(a : f) = K(f) - K(f|a)$. It is a natural indicator of the performance of a on f. Similarly, $K(a)$ measures the randomness of the algorithm a. Random instances are the "hardest" ones because they are incompressible and contain no internal structures for any algorithm to exploit. Random search algorithms are the least efficient algorithms because they convey no information about the problem and just visit the search space randomly. Given any f, can we deliberately design an a that performs *worse* than random search? If f happens to be a random one, we cannot design an algorithm to perform either better or worse than a random search. If it is a structured problem but we do not know any information about its structure, it is still hard for us to do so since we do not know what we should deceive. The only case we can do that is when f contains structural information one and we know what it is. The resulting algorithm can be called a *deceptively-solving algorithm* for its "purpose" is to seek deceptiveness and perform worse than random search. It is a structured algorithm since it contains structural information about the problem. But it uses this information in a "pathological" way. In this sense we can say a contains "negative information" about f. In the other hand, there are algorithms that are structured and perform better than random search. These algorithms can be called *straightforwardly-solving algorithms.* Therefore, there are three factors that cause an instance to be hard for a particular algorithm: *randomness*, *mismatch* and *deception.*

3 Misconceptions in GA-Hardness Research

There are three major misconceptions in the *problem space.* The first misconception is blurring the differences among the three above-mentioned instance hardness factors and feeding GAs with problems that are too hard to be meaningful. The second misconception is using a few, specific problem instances to support a general result. The third one is applying GAs on problems that are too general to be realistic. In the *algorithm space*, the most common misconception is considering GAs as a single algorithm and seeking a universal GA dynamics and general separation of GA-hard and GA-easy problems. For a given instance, if we change the parameters of GAs we can get totally different convergence results. So instead of taking GAs as a whole, researches into GA hardness should be done separately on different subclasses of GAs. The main misconception in the *performance space* is taking for granted the existence of a general *a priori* GA-hardness measure that can be used to predict a GA's performance on the given instance. Ideally, we want to have a program (a Turing machine) that can take as inputs a GA and an instance f of optimization problem and return how hard f is for the GA. Rice's theorem states that any non-trivial property of recursive enumerable languages is undecidable. This means that there are no nontrivial aspects of the behavior of a program which are algorithmically determinable given only the text of the program. It implies the nonexistence of general *a priori* predictive measure for problem hardness and algorithm performance based only on the description of the problem instance and the algorithm. This limits the general analysis, but it does not rule out the possibility of inducing an algorithm performance predictive model from some elaborately designed experimental results.

4 Future Directions: Experimental Approaches

We propose that future researches should focus more on real world NP-complete optimization problems rather than man-made functions; should study the classification of various GAs rather than considering them as a whole; and should give up analytically seeking a priori GA-hardness measures based *only* on the descriptive information of the problem and the GA, in favor of experimental, inductive approaches to learn the predictive model from the posterior results of running various GAs on problem instances with designed parameter settings.