A Guide For Fitness Function Design

Josh L. Wilkerson
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri, U.S.A.
jwilkerson@acm.org

Daniel R. Tauritz
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri, U.S.A.
dtauritz@acm.org

ABSTRACT

Fitness function design is often both a design and performance bottleneck for evolutionary algorithms. The fitness function for a given problem is directly related to the specifications for that problem. This paper outlines a guide for transforming problem specifications into a fitness function. The target audience for this guide are both non-expert practitioners and those interested in formalizing fitness function design. The goal is to investigate and formalize the fitness function generation process that expert developers go through and in doing so make fitness function design less of a bottleneck. Solution requirements in the problem specifications are identified and classified, then an appropriate fitness function component is generated based on its classifications, and finally the fitness function components combined to yield a fitness function for the problem in question. The competitive performance of a guide generated fitness function is demonstrated by comparing it to that of an expert designed fitness function.

Categories and Subject Descriptors: I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search

General Terms: Design, Algorithms

Keywords: Evolutionary Algorithm, Fitness Function Design, Fitness Function Classification

1. INTRODUCTION

The design of an effective fitness function for a given problem is often difficult, even for experienced designers. The goal of this research is to both create a guide to assist non-expert practitioners in the design of high performance fitness functions, and the formalization of fitness function design to provide a foundation for rigorous investigation.

The purpose of a fitness function is to guide the evolutionary process through the problem environment to an optimal solution. EA performance is strongly related to the quality of the fitness functions. The fitness function is the primary point in the EA where the problem specifications are enforced. For this reason, the problem specifications are an ideal location to start the fitness function design process. The presented guide starts by identifying the requirements that define a solution from the specifications. Each requirement is classified using the provided taxonomy and then a fitness function component is generated, based on the classifications applied, that is responsible for enforcing the requirement in the fitness function. These components are finally combined into a fitness function for the problem, which can be either composite or multi-objective.

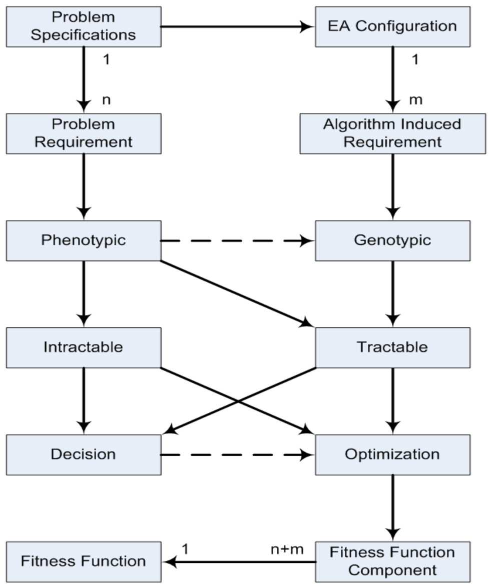

Figure 1: Requirement Classification Taxonomy

2. FITNESS FUNCTION GENERATION

In order to generate an effective fitness function, the characteristics of a valid solution to the problem must be defined. If properly indicated, the *Problem Specifications* should contain this information. Each solution requirement obtained from the problem specifications will ultimately yield a fitness function component. After the *Problem Requirements* have been identified, the next step is to begin bridging the gap between written problem requirements and a fitness function. The method proposed addresses this task by defining a taxonomy, which classifies the *Problem Requirements* and in doing so provides information on the nature of the fitness function. The taxonomy is shown in Figure 1.

Based on the given problem specifications, an appropriate solution representation and *EA Configuration* must be determined. There may be solution requirements that arise based on the algorithm selected. These *Algorithm Induced Requirements* arise due solely to the selected *EA Configuration* and as such cannot be expected to be included in the

Problem Specifications, since for a given problem there may be a number of applicable algorithms to use.

The first classification addresses how the requirement will be assessed. A *Phenotypic* requirement is based on some aspect of a candidate solution's expression in the problem environment, independent of the candidate's genetic representation. Conversely, a *Genotypic* requirement is based on some aspect of a candidate solution's genetic structure. Since the problem specifications are stated independent of a specific algorithm, it is impossible for a problem requirement to be *Genotypic*. Similarly, algorithm induced requirements are based solely on the algorithm selected and are independent of the specific problem being addressed and as such are always *Genotypic*. In many cases, it is advantageous to convert *Phenotypic* requirements to *Genotypic*, if possible. The reasoning behind this is that if desirable phenotypic behavior can be mapped to a genotypic configuration, then promoting this configuration will be much easier and convergence will occur much quicker.

The second classification is concerned with the practicality of assessing a given requirement. Essentially this classification determines if the resulting fitness function component will calculate the true fitness value (i.e., *Tractable*) or an approximation to the true fitness value (i.e., *Intractable*). *Tractble* requirements will operate on the entire problem domain for the requirement. *Intractable* requirements will operate on a sample set taken from the problem domain, which means a sampling method must also be decided upon for the fitness function component. Since the primary concern is practicality, a requirement may be classified as *Tractable* in one case and yet in another case, where fewer resources are available, the same requirement could be classified as *Intractable*. *Genotypic* requirements are based on the genetic structure of a candidate solution, which implies that the calculation of the true fitness of such a requirement should be feasible as long as the candidate solution representation is practical, thus these requirements are always *Tractable*.

The third classification defines the basic nature of the requirement. If a requirement is either satisfied or it is not, then it is a decision requirement. If there are intermediate levels of satisfaction of the requirement, then it is an optimization requirement. In order to create a graduated fitness function for the EA, any requirement that is classified as a decision requirement should be transformed into an optimization requirement. Additionally, the more gradient that each fitness function has, the better. So, some requirements may be classified as optimization, but will still need to be refined in order to generate a more effective fitness function.

The last step is to combine all of the fitness function components into the fitness function for the problem. One approach for this step is to combine the fitness function components into a single function in which the component fitness values are combined into a single fitness value. This option may work well for some cases; however, combining the various component fitness values can sometimes be difficult. A second approach is to use Multi-Objective EA [3] methods to calculate a fitness rank based on the Pareto Front generated by using each fitness function component as an objective.

3. GUIDE COMPARISON

[4] presents a comparison on eight buggy programs of the system presented by Arcuri et al. [2] with the CASC system. Results are presented on both CASC using a guide generated fitness function and using a close approximation of the fitness function used by Arcuri et al. The published results provide evidence that the guide generated fitness function performs at least as well as the Arcuri fitness function (and even performs better in some cases) for the first three bugs considered. For the fourth bug the guide fitness function is still competitive, though it does not perform as well as Arcuri's fitness function. In many of the CASC experiments using the Arcuri fitness function, one component of the fitness function dominated the others; similar observations were reported by Arcuri [1]. The guide generated fitness function, however, had no occurrences of a dominant fitness function component. The significance of these results is that they provide evidence that the guide can be used by a practitioner to generate a competitive fitness function (in terms of quality).

4. CONCLUSIONS AND FUTURE WORK

This paper presents a guide for fitness function design for non-expert practitioners as well as formalizes fitness function design, establishing a foundation for rigorous investigation of this critical EA component. A requirement classification taxonomy for fitness function design is presented which can, in a structured manner, be used to transform problem specifications into a set of fitness function components and ultimately their composition into a fitness function. Experiments are discussed that demonstrate the guide's ability to generate fitness functions that are competitive with expertly designed fitness functions. Future avenues of investigation for the guide include: inclusion of methods to determine the quality of fitness functions generated, development of tools to allow the use of advanced/specialized fitness function design techniques, extension of the guide to allow fitness function generation for black box search algorithms, and the usage of code templates to perform/assist in the implementation of fitness functions generated.

5. ACKNOWLEDGMENTS

This work was funded by the Missouri S&T Intelligent Systems Center.

6. REFERENCES

[1] A. Arcuri. *Automatic software generation and improvement through search based techniques.* PhD thesis, University of Birmingham, 2009.

[2] A. Arcuri and X. Yao. A novel co-evolutionary approach to automatic software bug fixing. In *Proceedings of IEEE CEC 2008*, pages 162-168, June 2008.

[3] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms.* John Wiley and Sons, 2001.

[4] J. Wilkerson and D. Tauritz. Coevolutionary Automated Software Correction. In *Proceedings of GECCO 2010*, pages 1391-1392, 2010.