GI4GI: Improving Genetic Improvement Fitness Functions

Mark Harman
University College London
Gower Street, WC1E 6BT
London, United Kingdom
mark.harman@ucl.ac.uk

Justyna Petke
University College London
Gower Street, WC1E 6BT
London, United Kingdom
j.petke@ucl.ac.uk

ABSTRACT

Genetic improvement (GI) has been successfully used to optimise non-functional properties of software, such as execution time, by automatically manipulating program's source code. Measurement of non-functional properties, however, is a non-trivial task; energy consumption, for instance, is highly dependant on the hardware used. Therefore, we propose the GI4GI framework (and two illustrative applications). GI4GI first applies GI to improve the fitness function for the particular environment within which software is subsequently optimised using traditional GI.

Categories and Subject Descriptors

D2.m [**Software Engineering**]: Miscellaneous

Keywords

GI, Genetic Improvement, Software Optimisation, Energy Optimisation, SBSE, Search Based Software Engineering

1. GI4GI FOR ENERGY OPTIMISATION

Recent work on Search Based Software Engineering (SBSE) has targeted energy optimisation; finding optimisations of software systems to reduce energy consumed, while retaining functionality [8, 9]. This work has used simulated annealing for tuning colour contrast parameters [8], and exhaustive search for tuning design patterns that minimise energy consumption [9], which might be thought of as a form of 'exhaustive search genetic improvement'.

The factors affecting energy consumption are many and varied, including screen behaviour, memory access, device communications and, of course, CPU utilisation. Since CPU utilisation is known to be correlated with energy consumption for GI [1, 12], existing approaches to genetic improvement that seek to reduce computational time [7, 11, 10, 13], will tend to also reduce energy consumption due to CPU utilisation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO'15, July 11-15, 2015, Madrid, Spain.

© 2015 ACM. ISBN TBA...$15.00

DOI: http://dx.doi.org/10.1145/2739482.2768415

However, in many situations, CPU utilisation is not the primary driver of energy consumption [8]. It is a challenge to determine which of the many other factors affect energy consumption, and their relative contribution. This is a challenge that could be addressed using genetic programming (GP): find an 'energy consumption equation' the free variables of which are the parameters that determine the ways in which devices, memory and screen are used. However, why evolve such an equation from scratch? For any given device, there is likely to be a wealth of information that would suggest candidate equations. Nevertheless, these candidates would not necessarily be particularly accurate. For a given device, application and user, they may need considerable tuning in order to provide reliable energy consumption predictions. Improving a given candidate could be viewed as a problem for genetic improvement.

Suppose we have used genetic improvement to find an improved equation, f, from a candidate equation, such that f predicts energy consumption in terms of a suitable space of input parameters, $\vec{x}$. We could use f as a fitness function to guide a second genetic improvement that targets the code itself, seeking to find an improved application that reduces energy consumed for a given set of test cases that determine the parameters, $\vec{x}$. In this way we use genetic improvement to find a fitness function to guide a subsequent genetic improvement that finds an energy efficient version of the given program or application. This is what we mean by Genetic Improvement for Genetic Improvement (GI4GI).

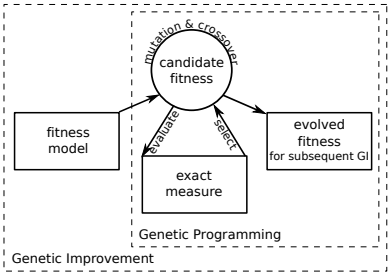

Figure 1: Phase 1 of GI4GI framework. The evolved fitness is used for genetic improvement of the target program (Phase 2). If fitness is evolved from scratch, the process can be viewed as GP for GI.

The idea of adapting the fitness function of an evolutionary algorithm has been proposed previously [2]. The novelty of our approach lies in using GI to achieve this task. A hardware-dependent linear energy model for GI has also been proposed [12]. However, the fitness function was not evolved, but obtained empirically for each piece of hardware.

2. GI4GI FOR SOFTWARE ARCHITECTURE OPTIMISATION

GI could also be applied to software architecture optimisation, targeting objectives such as throughput maximisation, response time minimisation, and another system performance characteristics which can be affected by architectural and design choices. Previous work on SBSE has targeted performance optimisation with considerable success [6]. Reformulating architectural performance optimisation as a GI problem may further increase the scope, and consequently performance improvements. While traditional SBSE approaches have the character of 'architectural parameter tuning', GI offers the opportunity to transform the architecture itself, not merely its parameters.

Computing fitness may require a simulation of the architecture and platform on which the software will be deployed. Such simulations can be computationally intensive. For example, Monte Carlo simulation may model usage scenario samples, but requires many individual simulation executions. It will be unrealistic to use such simulations as a direct component of the fitness function, invoked every time the fitness function is invoked.

Furthermore, one of the often claimed advantages of Search Based *Software Engineering* (compared to general engineering optimisation) is the way SBSE cuts out potential inaccuracies and miss-abstractions that occur through multiple layers of modelling and simulation [4]. In order to realise this SBSE advantage for software architecture optimisation, we would need to implement and execute the software architectures as the input fitness computation, rather than using simulations. However, this is unlikely to be computationally feasible, since it would likely consume even greater resources than any available simulation alternative.

GI4GI offers an alternative to using either simulation or actual architectures as the input of fitness computation. We propose a two phase process. In Phase 1, before starting any genetic improvement for architecture optimisation, we can first execute multiple instances of either simulation or actual architecture, in order to precompute a fitness function. We seek to find a formula for the fitness function that models the non-functional properties of interest sufficiently faithfully to be used in a subsequent guide to architectural performance GI, which would be performed in Phase 2.

If we have no pre-existing candidate formula for the fitness function, then Phase 1 is simply an application of GP. However, it seems reasonable to assume that there may be a candidate performance modelling equation that would form an initial suggestion. Running real instances in order to collect actual performance values could then be used to improve this generic fitness formula in Phase 1, improving its characterisation of the particular performance optimisation challenge to be attacked in Phase 2.

The pre-computation of the fitness function equation in Phase 1 does not avoid the problems associated with more general engineering optimisation (with their layers of modelling and simulation [4]). However, finding a good fitness equation may have value in its own right: it may yield insight into the performance drivers of the particular problem in hand and their relationships. Such insight is also one of the claimed advantages of SBSE [3], and GI could potentially start with previously reverse-engineered results [5]. Finding good fitness equation in Phase 1 may be just as valuable as finding a good architectural improvement in Phase 2. The architectural improvement achieved using GI with a fitness function found using GI could be seen as a way of evaluating the fitness function found in Phase 1, rather than the fitness function evaluating the architecture found in Phase 2.

3. REFERENCES

[1] B. R. Bruce, J. Petke, and M. Harman. Reducing energy consumption using genetic improvement. In 17^{th} *Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015. To appear.

[2] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[3] M. Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, pages 342–357. IEEE Computer Society Press, 2007.

[4] M. Harman. Why the virtual nature of software makes it ideal for search based optimization. In *Fundamental Approaches to Software Engineering,* 13^{th} *International Conference*, pages 1–12. Springer, 2010.

[5] K. Krogmann, M. Kuperberg, and R. Reussner. Using genetic search for reverse engineering of parametric behaviour models for performance prediction. *IEEE Transactions on Software Engineering*, 36(6):865–877, 2010.

[6] J. Kukunas, R. D. Cupper, and G. M. Kapfhammer. A genetic algorithm to improve linux kernel performance on resource-constrained devices. In 12^{th} *Annual Conference on Genetic and Evolutionary Computation*, pages 2095–2096. ACM, 2010.

[7] W. B. Langdon and M. Harman. Optimizing existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(1):118–135, 2015.

[8] D. Li, A. H. Tran, and W. G. J. Halfond. Making web applications more energy efficient for OLED smartphones. In 36^{th} *International Conference on Software Engineering*, pages 527–538. ACM, 2014.

[9] I. L. Manotas-Gutiérrez, L. L. Pollock, and J. Clause. SEEDS: a software engineer's energy-optimization decision support framework. In 36^{th} *International Conference on Software Engineering*, pages 503–514. ACM, 2014.

[10] M. Orlov and M. Sipper. Flight of the FINCH through the java wilderness. *IEEE Transactions on Evolutionary Computation*, 15(2):166–182, 2011.

[11] J. Petke, M. Harman, W. B. Langdon, and W. Weimer. Using genetic improvement and code transplants to specialise a C++ program to a problem class. In *Genetic Programming - 17^{th} European Conference*, pages 137–149. Springer, 2014.

[12] E. Schulte, J. Dorn, S. Harding, S. Forrest, and W. Weimer. Post-compiler software optimization for reducing energy. In 19^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 639–652. ACM, 2014.

[13] D. R. White, A. Arcuri, and J. A. Clark. Evolutionary improvement of programs. *IEEE Transactions on Evolutionary Computation*, 15(4):515–538, 2011.