ARTICLE IN PRESS
JID: INFSOF [m5G;September 11, 2017;12:1]

Information and Software Technology 000 (2017) 1–14

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Beyond evolutionary algorithms for search-based software engineering

Jianfeng Chen, Vivek Nair*, Tim Menzies

Department of Computer Science, North Carolina State University, Raleigh, NC, USA

ARTICLE INFO

Article history:
Received 25 January 2017
Revised 29 July 2017
Accepted 16 August 2017
Available online xxx

ABSTRACT

Context: Evolutionary algorithms typically require large number of evaluations (of solutions) to converge – which can be very slow and expensive to evaluate.
Objective: To solve search-based software engineering (SE) problems, using fewer evaluations than evolutionary methods.
Method: Instead of mutating a small population, we build a very large initial population which is then culled using a recursive bi-clustering chop approach. We evaluate this approach on multiple SE models, unconstrained as well as constrained, and compare its performance with standard evolutionary algorithms.
Results: Using just a few evaluations (under 100), we can obtain comparable results to state-of-the-art evolutionary algorithms.
Conclusion: Just because something works, and is widespread use, does not necessarily mean that there is no value in seeking methods to improve that method. Before undertaking search-based SE optimization tasks using traditional EAs, it is recommended to try other techniques, like those explored here, to obtain the same results with fewer evaluations.

1. Introduction

Due to the complexities of software architectures and shareholder requirements, it is often hard to solve complex modeling problems via a standard numerical mathematical analysis or some deterministic algorithms [25]. There are many reasons for this complexity:

- When procedural code is used within the model of a domain, every “if” statement can divide the internal problem space into different regions (once for each branch in the “if”). Such software models cannot be optimized via traditional numerical methods which assume models are a single continuous differentiable function.
- Finding solutions to problems often means accommodating competing choices. When stakeholders propose multiple goals, search-based SE (SBSE) methods can reflect on goal interactions to propose novel solutions to hard optimization problems such as configuring products in complex product lines [55], tuning parameters of a data miner [59], or finding best configurations for clone detection algorithms [63].

For these tasks, many SBSE researchers usually use evolutionary algorithms (EA) [55,59,63]. Evolutionary algorithms start by generating a set of initial solutions and improve them through crossover and mutation, also known as reproduction operators. They are inspired by evolution in nature and make no parametric assumptions about problems being generated. In our experience, this has made them particularly well-suited for SE problems. However, evolutionary algorithms typically require large number of evaluations (of solutions) to converge. Real-world model-based applications may be very expensive to evaluate (with respect to computation time, resources required etc.).

So, can we do better than EA for SBSE? Or, are there faster alternatives to EA? This paper experimentally evaluates one such alternative called SWAY (short for the Sampling WAY):

1. Similar to a standard EA, *generate* an initial population;
2. Intelligently *select* a cluster within the population generated with best scores.

SWAY runs so fast since it terminates after just $O(\lg N)$ evaluations of N candidate solutions. SWAY's intelligent selection mechanism for exploring subsets of the population is a recursive binary chop that (i) finds and evaluates only the two most dissimilar examples, then (ii) recurses only on half of the data containing the better among its similar example. As shown later in this paper, for this process to work, it is important to have the right definition of “dissimilar”.

* Corresponding author.
E-mail addresses: jchen37@ncsu.edu (J. Chen), vivekaxl@gmail.com, vnair2@ncsu.edu (V. Nair), tim.menzies@gmail.com, tim@menzies.us (T. Menzies).

http://dx.doi.org/10.1016/j.infsof.2017.08.007
0950-5849/© 2017 Published by Elsevier B.V.

Note the differences between SWAY and standard EA:

1. SWAY quits after the initial generation while EA reasons over multiple generations;
2. SWAY makes no use of reproduction operators so there is no way for lessons learned to accumulate as it executes;
3. Depending on the algorithm, not all members of the population will be evaluated – e.g. active learners [34] only evaluate a few representative individuals.

Because of the limited nature of this search, until recently, we would have dismissed SWAY as comparatively less effective than EA for exploring multi-goal optimization. Nevertheless, quite by accident, we have stumbled onto evidence that has dramatically changed our opinion about SWAY. Recently we were working with an algorithm called GALE [34]. GALE is an evolutionary algorithm that includes SWAY as a sub-routine:

$$evolution = generations * \begin{cases} mutation \\ crossover \\ sampling \end{cases}$$

$$SWAY = GALE - evolution$$

While porting GALE from Python to Java, we accidentally disabled evolution. To our surprise, the "broken" version worked as well, or better, than the original GALE. This is an interesting result since GALE has been compared against dozens of models in a recent TSE article [34] and dozens more in Krall's Ph.D. thesis [30]. In those studies, GALE was found to be competitive against widely used evolutionary algorithms. If Krall's work is combined with the results from our accident, then we conjecture that the success of GALE is due less to "evolution" than to "sampling" many options. This, in turn, could lead to a new generation of very fast optimizers since, as we show below, sampling can be much faster than evolving.

The rest of this paper describes SWAY and presents evidence for its utility. While we have tested SWAY on the standard EA benchmarks such as DTLZ, Fonseca, Golinski, Srinivas, etc. [16], those results are not included here since, in our experience, results from those benchmarks are less convincing to the SE community than results from software models. Hence, here we present results from:

- POM3: a model of agile teams selecting their next task from the scrum backlog [11,51];
- XOMO: a model predicting software development time, effort, risks and defects [41,44,45];
- MONRP: a model of next release planning that recommends which functionality to code next [4].

After presenting some background motivational notes, this paper offers general notes on multi-objective evolutionary algorithms. This is followed by a description of SWAY and the POM3, XOMO, MONRP models. Experimental results are then presented showing that SWAY achieves results competitive with standard methods (NSGA-II and SPEA2) using orders of magnitude fewer evaluations. Working with the MONRP models, we also find that a seemingly minor detail (the implementation of the distance function used to recognize "dissimilar" examples) is of vital importance to the success of SWAY. Finally, this paper concludes with experiments on "super-charging" that tests whether SWAY can boost the performance of standard optimizers.

Our observations after conducting the study are:

- The mutation strategies seen in a recently published EA algorithm (GALE) adds little value;
- GALE without evolution (SWAY) runs an order of magnitude faster than EAs;
- Optimization found by SWAY are similar to those found by SBSE algorithms;
- How we recognize "dissimilar" examples is of vital importance;
- Super-charging (combining SWAY with standard SBSE optimizers) is not useful.

More generally, our conclusion is that *sampling is an interesting research approach for multi-dimensional optimization that deserves further attention by the SBSE community.*

1.1. Connection to prior work

This paper significantly extends prior work by the authors. The background notes in the next section are new to the paper, as is the super-charging study. Also, this paper repairs a significant drawback seen in initial describing of SWAY. At SSBSE'16 [48], we demonstrated how SWAY can be used to find near optimal solutions for problems like XOMO and POM. While an interesting result, it turns out that the early definition of "dissimilar" used by the earlier version of SWAY was only applicable to problems whose decision space is constrained in nature. The results on other types of problems, were less than impressive. In this paper, we expose the weakness of the earlier variant of SWAY and show other definitions of "dissimilar" can make SWAY very useful for other domains.

1.2. When is SWAY most useful, useless?

SWAY is designed as a fast substitution of EAs for solving SBSE problems. It can avoid large amount of model evaluations, which are very common in previous evolutionary algorithms. In view of this, SWAY is particularly useful in following two scenarios.

SWAY would be most useful if it is proposed to put humans-in-the-loop to help guide the evaluations (e.g. as done in [53]). In this scenario, standard EAs might have to ask a human for up to $O(N^2)$ opinions for G generations. On the other hand, SWAY would only trouble the user $O(\lg N)$ times

Also, SWAY was created to solve problems, where the practitioner is not able to evaluate thousands of individuals for e.g. Wang et al. [64] spent 15 years of CPU time to find software clone detectors or model explored by Krall et al. [34] which take hours to perform a single evaluation.

However, as discussed later in this paper, SWAY has two core assumptions. Firstly, it is applicable only when there is a mapping between "genotype" and "phenotype" space; i.e. between the settings to the model inputs and outputs of the model. Even though such mapping may not exist in every model, we find here that for SE models (that were written with the explicit goal of effecting outputs with input decisions), this assumption holds adequately, at least for the purposes of improving model output.

Secondly, SWAY techniques for dividing the data makes the *spectral learning assumption*; i.e. that within the raw dimensions of data seen in any domain, there exists a small set of *spectral* dimensions which can usefully approximate the larger set [28]. While the universality of the spectral assumption has not been proven, it has seen to hold in many domains; e.g. see any data analysis method that uses principle components analysis [2,5,38,52,57].

1.3. Access to code

For the purposes of reproducibility, all the code and data used in this paper are available at http://tiny.cc/Sway.

2. Frequently asked questions

Before exploring the technical details on SWAY, we digress to answer some frequently asked questions about this research.