A review on Search-based Software Testing

Yu Zhang
Wuhan Texttile University
Student Number: 1815283001
Major: Software Engineering
Instructor: Peng Ye

Abstract—Search Based Software Testing (SBST) formulates testing as an optimisation problem, which can be attacked using computational search techniques from the field of Search Based Software Engineering (SBSE).

Search Based Software Testing refers to the use of meta-heuristics for the optimization of a task in the context of software testing. Meta-heuristics can solve complex problems in which an optimum solution must be found among a large amount of possibilities. The use of meta-heuristics in testing activities is promising because of the high number of inputs that should be tested. Previous studies on search based software testing have focused on the application of meta-heuristics for the optimization of structural and functional criteria. Recently, some researchers have proposed the use of SBST for mutation testing and explored solutions for the cost of application of this testing criterion.

Search based mutation testing is a field of interest, however, some issues remain unexplored. For instance, the use of meta-heuristics for the selection of effective mutation operators was identified in only one study. The results have pointed a range of possibilities for new studies to be developed, i.e., identification of equivalent mutants, experimental studies and application to different domains, such as concurrent programs.

Keywords: Search-based Software Testing (SBST), Search-based Software Engineering (SBSE), meta-heuristics, hyper-heuristics, Mutation Testing

I. Introduction

A. Search-based Software Engineering (SBSE)

Software Engineering (SE) often considers problems that involve finding a suitable balance between competing and potentially conflicting goals. There is often a bewilderingly large set of choices and finding good solutions can be hard. For instance, the following is an illustrative list of SE questions [1].

(1) What is the smallest set of test cases that covers all branches in this program?

(2) What is the best way to structure the architecture of this system to enhance its maintainability?

(3) What is the set of requirements that balances software development cost and customer satisfaction?

(4) What is the best allocation of resources to this software development project?

(5) What is the best sequence of refactoring steps to apply to this system?

Answers to these questions might be expected from literature on testing, design, requirements engineering, SE management, and refactoring, respectively. It might appear that these questions, which involve different aspects of software engineering, would be covered by different conferences and specialized journals and would have little in common. However, all of these questions are essentially optimization questions. As such, they are typical of the kinds of problem for which SBSE is well adapted and with which each has been successfully formulated as a search-based optimization problem. As we shall see in this survey, SBSE has been applied to testing, design, requirements, project management, and refactoring. This survey will show that work on SBSE applied to each of these five areas addresses each of the five questions raised before. This breadth of applicability is one of the enduring appeals of SBSE.

In SBSE, the term "search" is used to refer to the meta-heuristic Search-Based Optimization (SBO) techniques that are used. SBSE seeks to reformulate SE problems as SBO problems (or "search problems" for short). The use of the term "search" should not to be confused with "search" from other contexts such as textual or hypertextual search. Rather, for SBSE, a search problem is one in which optimal or near-optimal solutions are sought in a search space of candidate solutions, guided by a fitness function that distinguishes between better and worse solutions.

The interest in SBO for SE has led to an increased interest in other forms of optimization for SE that are not necessarily directly based on a "search". In the literature it is common to find the term "SBSE" applied to any form of optimization in which the problem domain comes from SE and the solution involves optimization according to some well-defined notion of fitness. In this article, we therefore include classical Operations Research (OR) techniques as well as metaheuristic "search-based" techniques in our understanding of SBSE.

It has been argued that the virtual nature of software makes it well suited for SBO. This is because fitness is computed directly in terms of the engineering artifact, without the need for the simulation and modeling inherent in all other approaches to engineering optimization. The field of SE is also imbued with rich metrics that can be useful initial candidates for fitness functions. This article aims to provide a comprehensive survey of SBSE. It presents research activity in categories drawn from the ACM subject categories within SE. For each, it lists the papers, drawing out common themes, such as the type of search technique used, the fitness definitions, and the

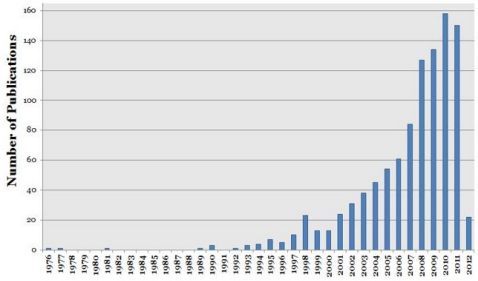

Fig. 1. increase in the quantity of SBSE research

nature of evaluation.

A wide range of different optimization and search techniques can and have been used. The most widely used are local search, Simulated Annealing (SA), Genetic Algorithms (GAs), Genetic Programming (GP), and Hill Climbing (HC). There is also increasing evidence of industrial interest in SBSE, with uptake by many software-centric organizations including Daimler, Ericsson, IBM, Microsoft, Motorola, Nokia, and NASA.

As the article reveals, 54% of the overall SBSE literature is concerned with SE applications relating to testing. There have been several important surveys in this widely studied general area. For this reason, the present survey will report overall trends in the wider SBSE literature(including Search-Based Testing), but it will defer to these other three surveys for details on the specific subfield of Search-Based Testing. The reader is also referred to an earlier (but considerably longer) version of this article that contains a detailed section on testing.

There has been a considerable increase in the quantity of SBSE research over the past few years (see Figure 1). Despite the excellent work in the surveys listed earlier, there remains, to date, no comprehensive survey of the whole field of study concerning trends in research. It is therefore timely to review the SBSE literature, the relationships between the applications to which it has been applied, the techniques used, trends, and open problems.

B. Search-based Software Testing (SBST)

Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing [2], [3]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces[4]. Test objectives find natural counterparts as the fitness functions used by SBSE to guide automated search, thereby facilitating SBSE formulations of many (and diverse) testing problems. As a result, SBST has proved to be a widely applicable and effective way of generating test data, and optimising the testing process. However, there are many exciting challenges and opportunities that remain open for further research and development, as we will show in this paper[5].

It is widely believed that approximately half the budget spent on software projects is spent on software testing, and therefore, it is not surprising that perhaps a similar proportion of papers in the software engineering literature are concerned with software testing. We report an updated literature analysis from which we observe that approximately half of all SBSE papers are SBST papers, a figure little changed since the last thorough publication audit (for papers up to 2009), which found 54% of SBSE papers concerned SBST [1]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [6], [7], [3], [8]. Therefore, rather than attempting another survey, we provide an analysis of SBST research trends, focusing on open challenges and areas for future work and development.

Since the first paper on SBST is also likely to be the first paper on SBSE, the early history of SBST is also the early history of SBSE. SBSE is a sub-area of software engineering with origins stretching back to the 1970s but not formally established as a field of study in its own right until 2001 , and which only achieved more widespread acceptance and uptake many years later.

The first mention of software optimisation (of any kind) is almost certainly due to Ada Augusta Lovelace in 1842. Her English language translation of the article (written in Italian by Menabrae), 'Sketch of the Analytical Engine Invented by Charles Babbage' includes seven entries, labelled 'Note A' to 'Note G' and initialed 'A.A.L'. Her notes constituted an article themselves (and occupied three quarters of the whole document). In these notes we can see perhaps the first recognition of the need for software optimisation and source code analysis and manipulation (a point argued in more detail elsewhere [9]):

"In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation." Extract from 'Note D'.

The introduction of the idea of software testing is probably due to Turing, who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both software testing and software verification. The first use of optimisation techniques in software testing and verification probably dates back to the seminal PhD thesis by James King, who used automated symbolic execution to capture path conditions, solved using linear programming. The first formulation of the test input space as a search space probably dates back seven years earlier to 1962, when a Cobol test data generation tool was introduced by Sauder. Sauder formulates the test generation problem as one of finding test inputs from a search space, though the search algorithm is random search, making this likely to be the first paper on Random

Test Data Generation. Sauder's work is also significant because it introduces the idea of constraints to capture path conditions, although these constraints are manually defined and not automatically constructed.

The first paper to use a meta-heuristic search technique was probably the work of Boyer, Elspas and Levitt on the SELECT system [10]. The paper is remarkable in many ways.

Here we can see, not only the first use of computational search (hill climbing) in software engineering, but also a hint at the idea (assignment of concrete values) that was subsequently to become Dynamic Symbolic Execution (DSE). Within this single paragraph we therefore may arguably find the origins of both DSE and SBST (and, by extension, SBSE too).

The SELECT paper is also remarkable in its sober and prescient assessment of the relative merits of testing and verification. Shortly after its publication, these two closely related research communities entered into a protracted and unhelpful 'feud' that generated a great deal more heat than light. Fortunately, we have more recently witnessed an accommodation between the two communities, and greater degree of welcome collaboration at their intersection. We really ought to ruefully reflect on the delay in this rapprochement given the 'understanding' already set out by the SELECT paper in 1975.

At about the same time2 Miller and Spooner [86], were also experimenting with optimisation-based approaches for generating test data (which they refer to as 'test selection' in the sense that they 'select' from the input space, which, in the more recent literature we would refer to as 'test data generation').

Unlike Boyer et al., Miller and Spooner used concrete execution of the program rather than symbolic execution, making their approach more similar to the techniques that ultimately became SBST, while the work of Boyer et al. followed a closely-related (but different) evolutionary path, which ultimately led to DSE. Current research develops both these techniques, and also hybrids that combine the best features of both [11], [12].

It appears that SBST research lay dormant for at approximately a decade until the work of Korel, which introduced a practical test data generation approach, the Alternating Variable Method (AVM), based on hill climbing. The first use of genetic algorithms for software engineering problems is usually attributed also to the field of SBST, with the work of Xanthakis et al., who introduced a genetic algorithm to develop whole test suites. Subsequent theoretical and empirical results tend to suggest that AVM outperforms genetic algorithms (in 'non-royal road' test data generation problems), at least for imperative programs in the C language. Since the late 1990s, with a greater overall software engineering focus on SBSE, there has been an explosion in SBST publications as the analysis below indicates.

Analysis of Trends in SBST: Figure 2 shows the growth

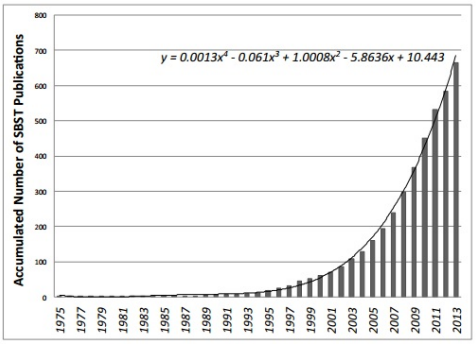

Fig. 2. Cumulative number of Search Based Software Testing papers. As can be seen, the overall trend continues to suggest a polynomial yearly rise in the number of papers, highlighting the breadth of interest and strong health of SBST.

in papers published on SBST. The data is taken from the SBSE repository. The aim of the repository is to contain every SBSE paper, underpinned by regular and careful human-based update. Although no repository can guarantee 100% precision and recall, the SBSE repository has proved sufficiently usable that it has formed the basis of several other detailed analyses of the literature, and is widely used by the SBSE community as a first source of information on related work. We found a close fit to a quartic function, indicating strong polynomial growth. If the trend continues, there will be more than 1,700 SBST papers before the end of this decade.

II. Conclusion

The future of SBSE is bright. The technology related to SBSE is certainly applicable to many fields, but it has not been fully considered. In the existing application field, the results have been very exciting.

If we think of software engineering as a real engineering discipline, then of course we should accept SBSE as a natural result.

Acknowledgment

Thanks for Professor Ye Peng's guidance and help in the past year. Now I have a more comprehensive understanding of software engineering course, and I have learned the knowledge that I missed before.

References

[1] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," ACM Comput. Surv., vol. 45, no. 1, Dec. 2012. [Online]. Available: https://doi.org/10.1145/2379776.2379787

[2] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," Information and Software Technology, vol. 51, no. 6, pp. 957 – 976, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584908001833

[3] P. McMinn, "Search-based software test data generation: a survey," Software Testing, Verification and Reliability, vol. 14, no. 2, pp. 105–156, 2004. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.294

[4] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," 2010.

[5] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015, pp. 1–12.

[6] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," IEEE Transactions on Software Engineering, vol. 36, no. 6, pp. 742–762, 2010.

[7] A. M. Mark Harman and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications." Technical, no. pp. 09-03, April 2009.

[8] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," Software Testing, Verification and Reliability, vol. 22, no. 2, pp. 67–120, 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.430

[9] M. Harman, "Why source code analysis and manipulation will always be important," in 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation, 2010, pp. 7–19.

[10] R. S. Boyer, B. Elspas, and K. N. Levitt, "Select—a formal system for testing and debugging programs by symbolic execution," in Proceedings of the International Conference on Reliable Software. New York, NY, USA: Association for Computing Machinery, 1975, p. 234–245. [Online]. Available: https://doi.org/10.1145/800027.808445

[11] A. Baars, M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, P. Tonella, and T. Vos, "Symbolic search-based testing," in 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011, pp. 53–62.

[12] K. Inkumsah and T. Xie, "Evacon: A framework for integrating evolutionary and concolic testing for object-oriented programs," in Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, ser. ASE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 425–428. [Online]. Available: https://doi.org/10.1145/1321631.1321700