Search Based Software Engineering

Mark Harman

King's College London, Strand, London, WC2R 2LS

Abstract. This paper was written to accompany the author's keynote talk for the Workshop on Computational Science in Software Engineering held in conjunction with International Conference in Computational Science 2006 in Reading, UK. The paper explains how software engineering activities can be viewed as a search for solutions that balance many competing constraints to achieve an optimal or near optimal result.
The aim of Search Based Software Engineering (SBSE) research is to move software engineering problems from human-based search to machine-based search, using a variety of techniques from the metaheuristic search, operations research and evolutionary computation paradigms. As a result, human effort moves up the abstraction chain to focus on guiding the automated search, rather than performing it. The paper briefly describes the search based approach, providing pointers to the literature.

1 Introduction

Software engineers often face problems associated with the balancing of competing constraints, trade-offs between concerns and requirement imprecision. Perfect solutions are often either impossible or impractical and the nature of the problems often makes the definition of analytical algorithms problematic.

Like other engineering disciplines, Software Engineering is typically concerned with near optimal solutions or those which fall within a specified acceptable tolerance. It is precisely these factors that make robust metaheuristic search-based optimization techniques readily applicable [33].

The growing international Search Based Software Engineering community has shown that search-based solutions using metaheuristic search techniques can be applied to software engineering problems right through the development lifecycle. For example, work has shown the applicability of search-based approaches to the 'next release' problem (requirements engineering) [5], project cost estimation [2, 10, 13, 14, 45], testing [7, 9, 16, 23, 24, 25, 58, 59], automated remodularisation (software maintenance) [29, 51], transformation [17, 18, 19, 6, 31] and studies of software evolution [8].

In exploring these applications, a range of search-based techniques have been deployed, from local search (for example [45, 51]) to genetic algorithms (for example [7, 29, 58, 59]) and genetic programming (for example [8, 14, 13, 16]). Techniques are also being developed to support search-based software testing by transforming software to assess [24] and improve [30] its evolutionary testability.

2 What is Search Based Software Engineering?

Search Based Software Engineering, as its name implies, treats software engineering problems as search problems, and seeks to use search techniques in order to solve the problems. Key to the approach is the re-formulation of a software engineering problem as a search problem [11, 27]. The term Search Based Software Engineering was coined in 2001 [33], since which time there has been a rapidly developing community working on this area with its own conferences and journal special issues. However, there was significant work on the application of search techniques to problems in software testing [20, 41, 42, 52, 55, 56, 60] and restructuring [15, 48] before the term 'Search Based Software Engineering' was coined to encompass the wider application of search to software engineering as a whole.

The search techniques used are a set of generic algorithms taken from the fields of metaheuristic search, operations research and evolutionary computation. These algorithms are concerned with searching for optimal or near optimal solutions to a problem within a large (possibly) multi-modal search space [21, 22, 40, 57].

For such problems, it is often infeasible to apply a precise analytic algorithm that produces the 'best' solution to the problem, yet it is possible to determine which is the better of two candidate solutions. Search techniques have been applied successfully to a number of engineering problems ranging from load balancing in the process industries (pressing of sugar pulp), through electromagnetic system design, to aircraft control and aerodynamics [61]. Search Based Software Engineering simply represents the application of these search algorithms to software engineering problems and the investigation of the implications of this novel application area.

Harman and Clark [27] identify four important properties in order for the Search Based Software Engineering approach to be successful:

1. **Large search space**
   If the fitness function is only used to distinguish a *few* individuals from one another, then the value of the fitness function for each individual can be computed and the search space explored exhaustively. There would be no need to use a search-based technique to sample the search space. Of course, most search spaces are *very* large. That is, most fitness functions apply to large (conceptually infinite) search spaces, such as the space of all expressible programs in some language or the space of all expressible designs in some design notation.
2. **Low computational complexity**
   Search based algorithms sample a portion of a very large search space. The portion sampled is typically non-trivial, requiring many thousands (possibly hundreds of thousands) of fitness evaluations. Therefore the computational complexity of the fitness function has a critical impact on the overall complexity of the search process. Fortunately, most fitness functions are relatively cheap to compute, since they can be constructed in terms of the structural or

syntactic properties of the programs, designs and systems which they assess and computed in time linear in the size of the program design or system.

3. **Approximate continuity**
   It is not necessary for a function to be continuous to be useful as a fitness function, but too much discontinuity can mislead a search, because all search–based optimisation approaches rely upon the guidance given by the fitness function. Continuity ensures that this guidance is perfect; the less continuous is the fitness function, the less guidance it gives.
4. **Absence of known optimal solutions**
   If there is a known optimal solution to a problem, then clearly there is no need to use a search-based approach to seek optimal (or near optimal) solutions.

Fortunately, these four problem characteristics are very prevalent in software engineering, where problems typically do involve a large search space (such as the number of possible designs, test cases or system configurations that may exist). Also, in many situations, there is no known optimal solution to the problem. The properties of 'low computational complexity' and 'approximate continuity' may not be present in all cases. However, even in cases where they are absent, it may be possible to transform the problem into one that is more amenable to Search Based Software Engineering [6, 31].

Interest in Search Based Software Engineering has grown rapidly in the past five years. For example, the work on search based testing is now sufficiently developed to merit its own survey paper [50], while there has been a healthy and growing Search Based Software Engineering track of the Genetic and Evolutionary Computation Conference GECCO, since 2002 and special issues and workshops on Search Based Software Engineering [26, 34].

3 Search Based Software Engineering Can Yield Fresh Insight

It has been widely observed that search techniques are good at producing unexpected answers. This happens because the techniques are not hindered by implicit human assumptions. One example is the discovery of a patented digital filter using a novel evolutionary approach [54]. Another example is the discovery of patented antenna designs [46] which are available commercially. The human formalises their (explicit) assumptions as a fitness function. Many of these are already available in the form of software metrics [27]. The machine uses this fitness function to guide the search. Should the search produce unexpected results then this reveals some **implicit** assumptions and/or challenges the human's intuition about the problem.

Unlike human–based search, automated search techniques carry with them no bias. They automatically scour the search space for the solutions that best fit the (stated) human assumptions in the fitness function. This is one of the central strengths of the approach. Software engineering is often hampered by poor human intuition and the presence of unstated and implicit assumptions. Automated search techniques will effectively work in tandem with the human, in an iterative process of refinement, leading to better fitness functions and, thereby, to better encapsulation of human assumptions and intuition.

Insight can also come from visualization of the landscape [39, 43, 44, 53]. That is, to use the fitness function values as a measure of height (or vertical co-ordinate), in a landscape where each individual in the search space potentially occupies some location within the horizontal co-ordinates.

Harman and Clark [27] describe other ways in which the SBSE approach can provide insight in the field of software metrics research, by providing a way to understand software metrics as fitness functions and to consider the effect of the metrics in terms of the optimizations that they produce when used as fitness functions.

4 Conclusion

Software engineering is essentially a search for a solution that balances many competing constraints to achieve an optimal or near optimal result. Currently, this search process is a highly labour–intensive human activity. It cannot scale to meet the demands of the new and emerging software engineering paradigms. Search Based Software Engineering addresses this problem head on, moving software engineering problems from human–based search to machine–based search. As a result, human effort will move up the abstraction chain, to focus on **guiding** the automated search, rather than **performing** the search itself.

5 Acknowledgements

This keynote arose as a result of recent work [1, 3, 4, 6, 11, 12, 17, 18, 19, 27, 28, 29, 30, 31, 32, 33, 35, 47, 49] undertaken by the author with many other colleagues in the growing Search Based Software Engineering community. The work is currently funded by a large EPSRC project, SEBASE, for which the other partners are John Clark (University of York) and Xin Yao (University of Birmingham) and industrialists from DaimlerChrysler Berlin, Motorola UK and IBM UK. This keynote draws on ideas from the SEBASE project and from other keynotes and tutorials prepared by the author in collaboration with Joachim Wegener at DaimlerChrysler [38, 37, 36].

References

1. Konstantinos Adamopoulos, Mark Harman, and Robert Mark Hierons. Mutation testing using genetic algorithms: A co-evolution approach. In *Genetic and Evolutionary Computation Conference (GECCO 2004), LNCS 3103*, pages 1338–1349, Seattle, Washington, USA, June 2004. Springer.

2. Jesús Aguilar-Ruiz, Isabel Ramos, José C. Riquelme, and Miguel Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 43(14):875–882, December 2001.
3. Giulio Antoniol, Massimiliano Di Penta, and Mark Harman. A robust search–based approach to project management in the presence of abandonment, rework, error and uncertainty. In 10^{th} *International Software Metrics Symposium (METRICS 2004)*, pages 172–183, Chicago, Illinois, USA, September 2004. IEEE Computer Society Press, Los Alamitos, California, USA.
4. Giulio Antoniol, Massimiliano Di Penta, and Mark Harman. Search-based techniques applied to optimization of project planning for a massive maintenance project. In 21^{st} *IEEE International Conference on Software Maintenance (ICSM 2005)*, pages 240–249, Budapest, Hungary, 2005. IEEE Computer Society Press, Los Alamitos, California, USA.
5. A.J. Bagnall, V.J. Rayward-Smith, and I.M. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, December 2001.
6. André Baresel, David Wendell Binkley, Mark Harman, and Bogdan Korel. Evolutionary testing in the presence of loop–assigned flags: A testability transformation approach. In *International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 108–118, Omni Parker House Hotel, Boston, Massachusetts, July 2004. Appears in Software Engineering Notes, Volume 29, Number 4.
7. André Baresel, Harmen Sthamer, and Michael Schmidt. Fitness function design to improve evolutionary structural testing. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1329–1336, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
8. Terry Van Belle and David H. Ackley. Code factoring and the evolution of evolvability. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1383–1390, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
9. Leonardo Bottaci. Instrumenting programs with flag variables for test data search by genetic algorithms. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1337–1342, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
10. Colin J. Burgess and Martin Lefley. Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology*, 43(14):863–873, December 2001.
11. John Clark, José Javier Dolado, Mark Harman, Robert Mark Hierons, Bryan Jones, Mary Lumkin, Brian Mitchell, Spiros Mancoridis, Kearton Rees, Marc Roper, and Martin Shepperd. Reformulating software engineering as a search problem. *IEE Proceedings — Software*, 150(3):161–175, 2003.
12. Karnig Derderian, Qiang Quo, Mark Harman, and Robert Hierons. Computing unique input/output sequences using genetic algorithms. In *3rd International Workshop on Formal Approaches to Testing of Software (FATES 2003)*, pages 164–177, Montréal, Canada, 2003. LNCS 2931.
13. Jose J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43(1):61–72, 1 January 2001.
14. José Javier Dolado. A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10):1006–1021, 2000.
15. D. Doval, S. Mancoridis, and B. S. Mitchell. Automatic clustering of software systems using a genetic algorithm. In *International Conference on Software Tools and Engineering Practice (STEP'99)*, Pittsburgh, PA, 30 August - 2 September 1999.

16. Maria Cláudia Figueiredo Pereira Emer and Silva Regina Vergilio. GPTesT: A testing tool based on genetic programming. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1343–1350, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
17. Deji Fatiregun, Mark Harman, and Rob Hierons. Evolving transformation sequences using genetic algorithms. In 4^{th} *International Workshop on Source Code Analysis and Manipulation (SCAM 04)*, pages 65–74, Chicago, Illinois, USA, September 2004. IEEE Computer Society Press, Los Alamitos, California, USA.
18. Deji Fatiregun, Mark Harman, and Rob Hierons. Search-based amorphous slicing. In 12^{th} *International Working Conference on Reverse Engineering (WCRE 05)*, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, November 2005. To appear.
19. Deji Fatiregun, Mark Harman, and Robert Hierons. Search based transformations. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 2511–2512, Chicago, 12-16 July 2003. Springer-Verlag.
20. Roger Ferguson and Bogdan Korel. The chaining approach for software test data generation. *ACM Transactions on Software Engineering and Methodology*, 5(1):63–86, January 1996.
21. F. Glover. Tabu search: A tutorial. *Interfaces*, 20:74–94, 1990.
22. David E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.
23. Hans G. Groß, Bryan F Jones, and David E Eyres. Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems. *IEE Proceedings Software*, (2):25–30, 2000.
24. Hans-Gerhard Groß. A prediction system for evolutionary testability applied to dynamic execution time. *Information and Software Technology*, 43(14):855–862, December 2001.
25. Hans-Gerhard Groß and Nikolas Mayer. Evolutionary testing in component-based real-time system construction. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 1393, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
26. Walter Gutjahr and Mark Harman. Focussed issue on search based software engineering. *Journal Computers and Operations Research*, 2006. To appear.
27. Mark Harman and John Clark. Metrics are fitness functions too. In 10^{th} *International Software Metrics Symposium (METRICS 2004)*, pages 58–69, Chicago, Illinois, USA, September 2004. IEEE Computer Society Press, Los Alamitos, California, USA.
28. Mark Harman, Chris Fox, Robert Mark Hierons, Lin Hu, Sebastian Danicic, and Joachim Wegener. Vada: A transformation-based system for variable dependence analysis. In *IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2002)*, pages 55–64, Montreal, Canada, October 2002. IEEE Computer Society Press, Los Alamitos, California, USA. Voted best paper by attendees.
29. Mark Harman, Robert Hierons, and Mark Proctor. A new representation and crossover operator for search-based optimization of software modularization. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1351–1358, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
30. Mark Harman, Lin Hu, Robert Hierons, André Baresel, and Harmen Sthamer. Improving evolutionary testing by flag removal ('best at GECCO' award winner). In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1359–1366, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

31. Mark Harman, Lin Hu, Robert Mark Hierons, Joachim Wegener, Harmen Sthamer, André Baresel, and Marc Roper. Testability transformation. *IEEE Transactions on Software Engineering*, 30(1):3–16, January 2004.
32. Mark Harman and Bryan Jones. SEMINAL: Software engineering using metaheuristic innovative algorithms. In 23^{rd} *International Confernece on Software Engineering (ICSE 2001)*, pages 762–763, Toronto, Canada, May 2001. IEEE Computer Society Press, Los Alamitos, California, USA.
33. Mark Harman and Bryan F. Jones. Search based software engineering. *Information and Software Technology*, 43(14):833–839, December 2001.
34. Mark Harman and Bryan F. Jones. The seminal workshop: Reformulating software engineering as a metaheuristic search problem. *Software Engineering Notes*, 26(6):62–66, November 2001.
35. Mark Harman, Stephen Swift, and Kiarash Mahdavi. An empirical study of the robustness of two module clustering fitness functions. In *Genetic and Evolutionary Computation Conference (GECCO 2005)*, Washington DC, USA, June 2005. Association for Computer Machinery. to appear.
36. Mark Harman and Joachim Wegener. Evolutionary testing. In *Genetic and Evolutionary Computation (GECCO)*, Chicago, July 2003.
37. Mark Harman and Joachim Wegener. Getting results with search–based software engineering. In 26^{th} *IEEE International Conference and Software Engineering (ICSE 2004)*, Edinburgh, UK, 2004. IEEE Computer Society Press, Los Alamitos, California, USA. To Appear.
38. Mark Harman and Joachim Wegener. Search based testing. In 6^{th} *Metaheuristics International Conference (MIC 2005)*, Vienna, Austria, August 2005. To appear.
39. E. Hart and P. Ross. GAVEL - a new tool for genetic algorithm visualization. *IEEE-EC*, 5:335–348, August 2001.
40. John H. Holland. *Adaption in Natural and Artificial Systems*. MIT Press, Ann Arbor, 1975.
41. B.F. Jones, H.-H. Sthamer, and D.E. Eyres. Automatic structural testing using genetic algorithms. *The Software Engineering Journal*, 11:299–306, 1996.
42. Bryan F. Jones, David E. Eyres, and Harmen H. Sthamer. A strategy for using genetic algorithms to automate branch and fault-based testing. *The Computer Journal*, 41(2):98–107, 1998.
43. Yong-Hyuk Kim and Byung-Ro Moon. Visualization of the fitness landscape, A steady-state genetic search, and schema traces. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 686, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
44. Yong-Hyuk Kim and Byung-Ro Moon. New usage of sammon's mapping for genetic visualization. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 1136–1147, Chicago, 12-16 July 2003. Springer-Verlag.
45. Colin Kirsopp, Martin Shepperd, and John Hart. Search heuristics, case-based reasoning and software project effort prediction. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1367–1374, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
46. D. S. Linden. Innovative antenna design using genetic algorithms. In D. W. Corne and P. J. Bentley, editors, *Creative Evolutionary Systems*, chapter 20. Elsevier, Amsterdam, The Netherland, 2002.
47. Kiarash Mahdavi, Mark Harman, and Robert Mark Hierons. A multiple hill climbing approach to software module clustering. In *IEEE International Conference*

on Software Maintenance (ICSM 2003), pages 315–324, Amsterdam, Netherlands, September 2003. IEEE Computer Society Press, Los Alamitos, California, USA.
48. Spiros Mancoridis, Brian S. Mitchell, C. Rorres, Yih-Farn Chen, and Emden R. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *International Workshop on Program Comprehension (IWPC'98)*, pages 45–53, Ischia, Italy, 1998. IEEE Computer Society Press, Los Alamitos, California, USA.
49. Phil McMinn, David Binkley, and Mark Harman. Testability transformation for efficient automated test data search in the presence of nesting. In *UK Software Testing Workshop (UK Test 2005)*, Sheffield, UK, September 2005.
50. Philip McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, June 2004.
51. Brian S. Mitchell and Spiros Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1375–1382, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
52. F. Mueller and J. Wegener. A comparison of static analysis and evolutionary testing for the verification of timing constraints. In *4th IEEE Real-Time Technology and Applications Symposium (RTAS '98)*, pages 144–154, Washington - Brussels - Tokyo, June 1998. IEEE.
53. Hartmut Pohlheim. Visualization of evolutionary algorithms - set of standard techniques and multidimensional visualization. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 533–540, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.
54. T. Schnier, X. Yao, and P. Liu. Digital filter design using multiple pareto fronts. *Soft Computing*, 8(5):332–343, April 2004.
55. N. Tracey, J. Clark, and K. Mander. Automated program flaw finding using simulated annealing. In *International Symposium on Software Testing and Analysis*, pages 73–81. ACM/SIGSOFT, March 1998.
56. Nigel Tracey, John Clark, Keith Mander, and John McDermid. Automated test-data generation for exception conditions. *Software Practice and Experience*, 30(1):61–79, 2000.
57. P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Practice*. Kluwer Academic Publishers, Dordrecht, the Netherlands, 1987.
58. Joachim Wegener, André Baresel, and Harmen Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms*, 43(14):841–854, 2001.
59. Joachim Wegener and F. Mueller. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Real-Time Systems*, 21(3):241–268, 2001.
60. Joachim Wegener, Harmen Sthamer, Bryan F. Jones, and David E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality*, 6:127–135, 1997.
61. G Winter, J Periaux, M Galan, and P Cuesta. *Genetic Algorithms in Engineering and Computer Science*. Wiley, 1995.