Getting Results from Search–Based Approaches to Software Engineering

Mark Harman,
Brunel University,
Uxbridge, Middlesex,
UB8 3PH, UK.
Mark.Harman@brunel.ac.uk

Joachim Wegener,
DaimlerChrysler,
Alt-Moabit 96a,
D-10559 Berlin, Germany.
Joachim.Wegener@DaimlerChrysler.com

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. Software engineering problems often lead to observations such as those below:

"We need to balance **competing constraints**."
"We have to cope with **inconsistency**."
"Unfortunately there are **many potential solutions**."
"There is **no perfect answer**..."
"...but I **can distinguish good ones from bad**."

It is precisely these observations which make robust, meta-heuristic, search-based optimization techniques readily applicable. For the past twenty years, engineers from the fields of mechanical, chemical, electrical and civil engineering have been applying search–based techniques, such as simulated annealing and genetic algorithms, to arrive at optimal and near optimal solutions to constrained problems within large search spaces.

More recently, search–based techniques has started to find application in *software engineering* problem domains. This area of *search–based* software engineering has its origins in work on search–based testing, which began in the mid 1990s. In the past four years the field has experienced a rapid increase in activity; there are now over fifty universities and industrial research/practitioner groups working on search–based software engineering, and the spectrum of new results and application areas continues to grow rapidly.

Already, search-based solutions have been applied to software engineering problems right through the development life-cycle. For example, existing work has shown the applicability of search-based approaches to, among others, the 'next release' problem (requirements engineering) [2], project cost estimation [1, 6, 8, 9, 16], testing [3, 5, 12, 20, 22, 23, 24, 25] automated re-modularisation (software maintenance) [10, 13, 17, 18, 19, 21], transformation [11, 14, 26] and studies of software evolution [4]. An introductory overview of the field can be found in [15], while a more detailed survey can be found in [7].

The tutorial will provide each participant with the ability to exploit Search–Based Software Engineering (SBSE) theory and techniques and to be able to apply them to a chosen area of software engineering.

The objectives are that, having taken the tutorial, each participant will be able to:

1. Have a working understanding of two key search techniques: Genetic Algorithms and Hill Climbing.

2. Understand the three crucial ingredients; Representation, Fitness function and optimization technique.

3. Understand how to determine whether the application of these techniques is effective.

4. Have an appreciation of some of the advanced topics in SBSE: Convergence/Stopping criteria, Hybrid search approaches, Search space reduction and Fitness landscape transformation.

Search-Based software Engineering is a fast growing field. Growth in interest is fueled by the way in which search techniques can be applied right across the life-cycle and the speed with which the techniques can be mastered and deployed to produce results.

References

[1] J. Aguilar-Ruiz, I. Ramos, J. C. Riquelme, and M. Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 43(14):875–882, Dec. 2001.

[2] A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, Dec. 2001.

[3] A. Baresel, H. Sthamer, and M. 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.

[4] T. V. Belle and D. 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.

[5] L. 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.

[6] C. J. Burgess and M. Lefley. Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology*, 43(14):863–873, Dec. 2001.

[7] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating software engineering as a search problem. *IEE Proceedings — Software*, 150(3):161–175, 2003.

[8] J. J. Dolado. A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10):1006–1021, 2000.

[9] J. J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43(1):61–72, 1 Jan. 2001.

[10] 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.

[11] D. Fatiregun, M. Harman, and R. 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.

[12] H.-G. Groß. A prediction system for evolutionary testability applied to dynamic execution time. *Information and Software Technology*, 43(14):855–862, Dec. 2001.

[13] M. Harman, R. Hierons, and M. 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.

[14] M. Harman, L. Hu, R. M. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper. Testability transformation. *IEEE Transactions on Software Engineering*, 30(1):3–16, Jan. 2004.

[15] M. Harman and B. F. Jones. Search based software engineering. *Information and Software Technology*, 43(14):833–839, Dec. 2001.

[16] C. Kirsopp, M. Shepperd, and J. 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.

[17] K. Mahdavi, M. Harman, and R. M. Hierons. A multiple hill climbing approach to software module clustering. In *IEEE International Conference on Software Maintenance (ICSM 2003)*, pages 315–324, Amsterdam, Netherlands, Sept. 2003. IEEE Computer Society Press, Los Alamitos, California, USA.

[18] S. Mancoridis, B. S. Mitchell, Y.-F. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings; IEEE International Conference on Software Maintenance*, pages 50–59. IEEE Computer Society Press, 1999.

[19] S. Mancoridis, B. S. Mitchell, C. Rorres, Y.-F. Chen, and E. 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.

[20] C. Michael, G. McGraw, and M. Schatz. Generating software test data by evolution. *IEEE Transactions on Software Engineering*, (12):1085–1110, Dec. 2001.

[21] B. S. Mitchell and S. 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.

[22] R. P. Pargas, M. J. Harrold, and R. R. Peck. Test-data generation using genetic algorithms. *The Journal of Software Testing, Verification and Reliability*, 9:263–282, 1999.

[23] M. Roper. Cast with gas (genetic algorithms) - automatic test data generation via. evolutionary computation. In *IEE Colloquium on Computer Aided Software Testing Tools*. IEE, April 1996.

[24] J. Wegener, A. Baresel, and H. 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.

[25] J. 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.

[26] K. P. Williams. *Evolutionary Algorithms for Automatic Parallelization*. PhD thesis, University of Reading, UK, Department of Computer Science, Sept. 1998.