Information and Software Technology 43 (2001) 833–839

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Search-based software engineering

Mark Harman[a,*], Bryan F. Jones[b,1]

[a]Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
[b]School of Computing, University of Glamorgan, Pontypridd, CF37 1DL, UK

Abstract

This paper claims that a new field of software engineering research and practice is emerging: search-based software engineering. The paper argues that software engineering is ideal for the application of metaheuristic search techniques, such as genetic algorithms, simulated annealing and tabu search. Such search-based techniques could provide solutions to the difficult problems of balancing competing (and some times inconsistent) constraints and may suggest ways of finding acceptable solutions in situations where perfect solutions are either theoretically impossible or practically infeasible.

In order to develop the field of search-based software engineering, a reformulation of classic software engineering problems as search problems is required. The paper briefly sets out key ingredients for successful reformulation and evaluation criteria for search-based software engineering. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Software engineering; Metaheuristic; Genetic algorithm

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 which make robust metaheuristic search-based optimisation techniques readily applicable.

Metaheuristic algorithms, such as genetic algorithms (GA) [17], simulated annealing [37] and tabu search [16] have been applied successfully to a number of engineering problems. For example, a literature survey of genetic algorithms reveals applications to, among others:

- mechanical engineering [24,33]
- chemical engineering [8,22]
- medical and biomedical engineering [29,32,40]
- civil engineering [1,6,14,19]
- electronic engineering [5,11,25]

GA research and researchers have even received interest from observers in the field of social science. Though GA practitioners may not agree with the findings of sociologists [18], it is an indication of the wide appreciation of the significance of these search-based technologies that they should have penetrated the collective consciousness of even 'non-technical' disciplines such as social science.

However, the discipline of software engineering appears to be unique with regard to the application of genetic algorithms (and similar search-based, metaheuristic optimisation techniques); metaheuristic algorithms have received comparatively little attention from software engineers in comparison with that which they have received from researchers and practitioners in the more established fields of engineering.

A literature search on 'software engineering' and 'genetic algorithms' reveals work within the areas of testing [20,21,36,28,38,39] and cost estimation [12,13], but few others. Work has also taken place on automatic programming using genetic programming [23] and parallelisation [31,44], which could be thought of specific topics within coding, which in turn, could be considered to be a part of software engineering. Even with the inclusion of this work on coding, the application of search-based techniques to problems in software engineering has been, hitherto, somewhat patchy. The thesis underpinning the present paper is that search-based metaheuristic optimisation techniques are highly applicable to software engineering and that their

* Corresponding author. Tel.: +44-1895-274-000; fax: +44-1895-251-686.

E-mail addresses: mark.harman@brunel.ac.uk (M. Harman), bfjones@glam.ac.uk (B.F. Jones).

[1] Tel.: +44-1443-482730; fax: +44-1443-482715

investigation and application to software engineering is long overdue. It is time for software engineering to catch up with its more mature counterparts in traditional fields of engineering.

Software engineering problems are often typified [4,3,10,27,26,30,35] by the observations paraphrased below:

There is usually a need to balance *competing constraints*.
Occasionally there is a need to cope with *inconsistency*.
There are often *many potential solutions*.
There is typically *no perfect answer*… but *good ones* can be *recognised*.
There are sometimes *no precise rules for computing the best solution*.

These properties of software engineering are also found to be in other engineering disciplines and they are *precisely* the attributes which make the application of search-based techniques so attractive. In order to apply metaheuristic search-based techniques to software engineering, the problems software engineers face must be reformulated as search problems. In Section 2, the paper briefly sets out the key ingredients for such a reformulation. Section 3 provides some criteria against which to judge the success (or otherwise) of an application of search-based techniques to a software engineering problem. This paper is essentially a manifesto for search-based software engineering. Section 4 sets out some research goals for search-based software engineering, while Section 5 concludes with a call for more work in this area.

2. Reformulating software engineering as a search problem

The principal intention of this section is to demonstrate how conceptually simple is the reformulation of software engineering to search-based software engineering. It is hoped that the reader is convinced that, at least in principal, it will be possible to apply metaheuristic search to a large body of software engineering problems, where natural representations, fitness functions and operators suggest themselves.

In order to reformulate software engineering as a search problem, it will be necessary to define:

- a representation of the problem which is amenable to symbolic manipulation,
- a fitness function (defined in terms of this representation) and
- a set of manipulation operators.

In addition, the perception of the problem itself may need to shift, in order to recast it as a search-based problem. For example, in the case of testing, the problem becomes one of searching for test cases, which satisfy some test-adequacy criterion.

These issues are covered in far more detail in the general literature on metaheuristic search [17,41,43]. They are only mentioned here to demonstrate that the reformulation suggested and consequent development of search-based software engineering is conceptually feasible.

2.1. Representation

The representation of a candidate solution is critical to shaping the nature of the search problem. Representations, which are frequently used in existing applications for problem parameters are floating point numbers and binary code. In the latter case, grey codes are generally preferred to ‘pure binary’ numbers [42], since successor decimals (for instance 7 and 8) are not near neighbours in a pure binary code (4 mutations are required to mutate the four differing digits, rather than one). Grey coding avoids this problem.

2.2. Fitness

The fitness function is the characterisation of what is considered to be a good solution. In measurement theoretical terms [34], the fitness function need merely impose an ordinal scale of measurement upon the individual solutions it is applied to. That is, it will generally be sufficient to know which of two candidate solutions is the better according to the properties (or set of properties) to be measured. The fitness function imposes a fitness landscape, the characteristics of which will both determine which search techniques are most applicable and will shed light on the nature of the problem and its candidate solutions in terms of their perceived fitness. Fitness landscapes should not be too flat, nor should they have sharp maxima that can easily be missed. Often the fitness function, which first occurs, requires tuning to avoid these problems and to help guide the search towards good solutions.

In some cases it may be hard to define a fitness function, because the artefact to be optimised may have aesthetic qualities which make the determination of an ordinal scale metric difficult. However, in such situations a search-based approach may still be applicable, as will be argued in Section 3.3.

2.3. Operators

Different search techniques use different operators. As a minimum requirement, it will be necessary to mutate an individual representation of a candidate solution to produce a representation of a different candidate solution. Clearly, this is a very minimal requirement. It will make it possible to apply hill climbing approaches and certain forms of evolutionary computation. If it is also possible to determine the set of ‘near neighbours’ of a candidate solution (in terms of its representation) then simulated annealing and tabu search can be applied. If instead (or in addition), it is possible to sensibly cross-over two individuals (to produce a 'child' which retain characteristics of both 'parents') then genetic algorithms will be applicable.

This section has glossed over a lot of issues which underly the choice of representation, fitness function and operators. The intention was not to suggest that these issues are unimportant. On the contrary, such issues are likely to form the large body of research work to be undertaken to successfully move from speculation to application. However, this work will be highly domain dependent and so cannot be adequately covered in this overview paper.

3. Evaluation criteria for search-based software engineering

This section sets out some guidelines against which an attempt to apply metaheuristics to software engineering can be validated. The base line criteria suggest essential goals for any search-based approach. For a more thorough validation, Section 3.2 suggests ways in which search-based software engineering might be combined with classical software engineering and the ways in which these combined approaches might be validated.

3.1. Base line validity

To achieve a measure of base line acceptability, a metaheuristic technique must out-perform a purely random search. That is, metaheuristics should find better solutions or find them with less computational effort than random search. Metaheuristic techniques should be able to find values, which compare well with known solutions and should tend to produce fitness values, which are considered acceptable for a reasonable subclass of elements of the problem space. Of course, these latter two criteria are somewhat vague and depend upon the application domain. This section merely sets out some guidelines.

3.1.1. Random search

Any search-based technique must perform better than a purely random search in order to qualify as worthy of consideration as a successful application of search-based software engineering. Random search therefore provides a lowest benchmark for the application of metaheuristic search. Where a metaheuristic search does not consistently outperform random search it must be rejected. However, where this occurs, it may well indicate a poor choice of search technique (for example hill climbing in a multimodal 'rugged' fitness landscape), and therefore may invalidate merely the choice of technique or its implementation, rather than the whole notion of applying metaheuristics to the software engineering problem under consideration.

3.1.2. Discovery of known solutions

Even in situations where there is no known general algorithm for solving a problem, there may be known examples, constructed by hand, where individual solutions are known for particular elements of the problem domain. One natural approach to validation is, therefore, to establish whether the metaheuristic search is able to find solutions that compare well with these known individual solutions.

Of course, 'compare well' requires some elucidation. If the known solutions are provided for a small set of simple problems, then metaheuristic search should perform as well as or better. However, where hand constructed solutions are used to illustrate 'difficult problems' for which solutions are hard to define, the phrase 'compare well' might be interpreted less stringently.

3.1.3. Discovery of desirable solutions

Since the fitness function is a measure of 'goodness' of the solutions, another natural approach to validation consists of seeing just how much the fitness increases and gathering empirical data to provide evidence of the kinds of solution which may be obtained for typical problem elements.

3.1.4. Efficiency

In many (but by no means all) cases, a search-based approach may be slower than an existing analytical approach, because the search will involve repeated trials of the fitness function to evaluate candidate solutions. However, so long as the search-based technique can produce better solutions, there will be many software engineering applications where the search-based approach is more appropriate because quality overrides speed.

Where there exists a technique for producing solutions which is always applicable and which produces consistently high-quality solutions, a search based approach will clearly still be preferable if it produces solutions of equal or better quality more speedily.

3.2. Validation with respect to existing analytical techniques

In this section, the phrase 'analytical techniques' is used to describe any non-search-based algorithm for constructing solutions to the software engineering problem in question.

Where there exist analytic techniques for constructing reasonable solutions, search-based techniques will still be applicable. For instance, where analytical techniques are known to favour certain forms of solution or to have biases, which affect their behaviour, a search-based technique may find solutions, which cannot be discovered by the analytical technique. Where analytical techniques are applicable only to a subset of the problem space, search techniques may provide a mechanism for 'filling in the gaps'. Where analytical techniques are not consistently good at constructing solutions, search-based techniques may be used as a 'second guess'. Finally, where analytic techniques exist, but are known to produce sub-optimal solutions, these existing techniques may be used to seed the metaheuristic search with an initial population of reasonable solutions. It would then be hoped that the metaheuristic search would improve upon these initial seeds.

For each of these situations, analytic techniques produce only a part of the overall answer and metaheuristic search can be used to augment the existing approach.

3.2.1. Avoiding bias in analytical techniques

Where metaheuristic search is used to address bias in existing analytic solutions, the evaluation criteria will measure the novelty of solutions relative to those produced (and their relative fitness) compared to those produced by the analytic approaches. In such situations, even where the metaheuristic technique does not outperform the analytical techniques on fitness, it may be validated in terms of the additional insight gained from considering solutions previously thought atypical or non-standard.

3.2.2. Filling problem space gaps left by analytical solutions

Where existing analytic techniques are not applicable to the entire problem space, the use of metaheuristics may be partially validated, simply by showing that they ‘fill in the gaps’. However, a stronger validation would clearly involve demonstrating comparable (or improved) fitness of metaheuristic solutions relative to those produced by analytic algorithms. However, where problem subspaces uncovered by analytic algorithms represent ‘harder’ problems to solve than those which are covered, this criterion may be too stringent. In this situation, validation may simply show that the metaheuristic technique is capable of producing better solutions for previously uncovered areas of the problem space than can be found with random search alone.

3.2.3. Optimisation of partial or suboptimal analytical solutions

Where metaheuristic techniques are used to provide second guesses or where such algorithms are seeded by an initial population of results from an existing analytic algorithm, the metaheuristic technique should clearly be shown to provide improvement (i.e. increased fitness).

A simple-minded hybrid approach could simply try both the existing approach and a search-based approach and select the best solution. Fortunately, any such combined analytical and metaheuristic technique need only produce better results than the pure analytical approach on ‘a few’ occasions in order to be considered worth while. The interpretation of ‘a few’ will depend upon the domain of application.

3.2.4. Choice of technique and fitness function

The general application of metaheuristic techniques to software engineering presents a large number of choices. The representation, fitness functions, operators and search technique to apply must all be carefully considered. For some applications, a simple hill-climbing approach may produce either good or adequate results. For other problems, the multi-modal nature of the fitness landscape may demand a more robust technique. In general, it will be wise to experiment with a variety of techniques, fitness functions and operators. Fortunately, the highly generic nature of search-based techniques supports just this form of experimentation.

Finally, for some applications, the fitness function itself may require some form of validation. Where there exist well-understood and widely accepted metrics for assessing elements of the solution space, the fitness function will not require validation; it can simply be appropriated from the software measurement literature. However, in the software measurement literature there are few uncontroversial metrics [34,15]. Where there is disagreement about how to measure candidate solutions, the fitness function used and the landscape it imposes on the solution space will require validation. In all cases, the algorithmic complexity of the computation of the fitness function may be an issue. In some application areas, a computationally demanding fitness evaluation may render the approach impractical, even where searches are often highly productive with respect to the solutions found.

3.3. Psychological considerations

Software engineering involves aesthetic as well as technical qualities. It may be found, as with more aesthetic applications of metaheuristic search [7,9,2], that a non-automatic, human-based fitness evaluation is required. In such cases, the psychological implications of repeated trials on the same human subject (and the judgment biases that this may induce) should be considered.

There is also a ‘human dimension’ to search-based software engineering research where the solution produced requires an explanation. In most cases, the fitness function should be well-understood, so that the solutions can be explained purely in terms of their superior fitness. However, it may also be the case that the metaheuristic search approach is applied to give insight into the fitness function itself. Where a metric which captures the property (or properties) of interest is hard to define, the search may be used to explore the fitness landscapes created by several candidate metrics/fitness functions.

4. Future work

The authors expect to see a dramatic development of the field of search-based software engineering. The special issue of Information and Software Technology, of which this paper forms a part, represents the embryonic state of a rapidly growing sub-area in software engineering research. This section sets out some goals and milestones for the emergent search-based software engineering research community.

It seems likely that within five years, metaheuristic search will have been applied to, at least, the following areas of software engineering. (To the authors' knowledge, work is already in progress in several of these areas.)

1. Requirements prioritisation
2. Finding good designs
3. Test data selection
4. Reverse and re-engineering through transformation and re-factoring
5. Development of software measurement
6. User-based fitness evaluation for aesthetic aspects of software engineering

In addition, it is hoped that the field of search-based software engineering will provide:

1. Comparative studies and evaluations of different techniques as applied to software engineering problems. In an embryonic field of research, it is natural that many results will represent isolated advances in related but slightly different directions. As the field develops and matures, it is reasonable to hope for and expect that comparative studies and surveys will start to appear. The relative number of such studies might be thought of as a crude first approximation to an assessment of the maturity of search-based software engineering.
2. Theoretical development, in order to answer questions such as 'which technique/operators are likely to be fruitful for software engineering problem X?'. Comparative studies will help to answer questions like 'which technique works best for problem X?'. In addition, it is to be hoped that development of search-based software engineering will be deep as well as broad; that theoretical studies will emerge to complement the empirical work in the field.
3. Analyses of fitness landscapes for software engineering problems. This may yield deeper insight into the nature of the problems themselves.
4. Tailoring of fitness functions. Examination of the behaviours of metaheuristic search, may suggest modifications to the fitness landscape in order to improve the success of the search. Such modifications may produce useful feedback to the software measurement community. In any case, the wide-spread development of search-based software engineering is likely to inform and be informed by work on software measurement, since there is a clear and strong connection between software metrics and fitness evaluation.
5. Deeper appreciation of metaheuristics within the wider software engineering community. Currently, many software engineers are only tangentially familiar with metaheuristic search. It is hoped that the reformulation of many software engineering problems as search-based problems will widen awareness of metaheuristic search within the wider software engineering community.
6. Wider application of search-based software engineering in practice. Naturally, it is hoped that the next five years will witness a large increase in the application of metaheuristic search, building on the existing successes with test-data generation.
7. Provision of surprising solutions. One of the novel features which search-based software engineering adds to software engineering is the way in which the solutions found are occasionally surprising and not previously considered by analytic solutions. The metaheuristic search simply aims to optimise the value of the fitness function. Such a search-based approach is therefore entirely declarative; it is formulated in terms of what is required rather than how it is to be constructed. In other applications of metaheuristic search, it has been observed that this declarative nature of the approach leads to unexpected solutions which tend to yield additional insight into the formulation of the problem.

5. Conclusion

Software engineering problems are, by their very nature, ideal for the application of metaheuristic search techniques. This paper has argued that a new field of software engineering research, search-based software engineering is emerging. This sub-area of software engineering seeks to reformulate software engineering problems as search-based problems, facilitating the application of metaheuristic search-based techniques such as genetic algorithms, simulated annealing and tabu search.

The paper sets out key ingredients for this reformulation and provides criteria against which the success of individual approaches and the development of the field of research can be measured. The authors believe that the development of search-based software engineering is long overdue. The aim of this paper (and part of the motivation for this special issue) is to encourage a 'research gold rush'. Many exciting and challenging problems in software engineering lie within reach of metaheuristic search.

Acknowledgements

This paper is essentially a manifesto for search-based software engineering. The authors have benefited from extensive discussion within the auspices of the EPSRC-funded SEMINAL[2] network (GR/M78083). In particular, the authors would like to acknowledge the contribution made by Len Bottaci, Colin Burgess, John Clark, Jose Dolado, Gerdi Gross, Rob Hierons, Martin Lefley, Rudi Lutz, Mark Proctor, Vic Rayward Smith, Marc Roper, Nick Sharples, Martin Shepperd, Harman Sthamer, Nigel

[2] Software Engineering using Metaheuristic INnovative ALgorithms.

Tracey, Joachim Wegener and Darrell Whitley. Of course, all errors and omissions remain the sole responsibility of the authors.

References

[1] V. Babovic, Mining sediment transport data with genetic programming, Proceedings of the First International Conference on New Information Technologies for Decision Making in Civil Engineering (Montreal, Canada, 11–13 October 1998) 1998 pp. 875–886.

[2] S. Baluja, D. Pomerleau, T. Jochem, Simulating user's preferences: towards automated artificial evolution for computer generated images, Technical Report CMU-CS-93-198, Carnegie Mellon University, Pittsburgh, PA, October 1993.

[3] R. Balzer, Tolerating inconsistency (software development), Proceedings of the 13th International Conference on Software Engineering 1991 pp. 158–165.

[4] N.S. Barghouti, B. Krishnamurthy, Using event contexts and matching constraints to monitor software processes effectively, Proceedings of the 17th International Conference on Software Engineering, ACM Press, New York, 1995 pp. 83–92.

[5] F.H. Bennett III, M.A. Keane, D. Andre, J.R. Koza, Automatic synthesis of the topology and sizing for analog electrical circuits using genetic programming, in: K. Miettinen, M.M. Mäkelä, P. Neittaanmäki, J. Periaux (Eds.), Evolutionary Algorithms in Engineering and Computer Science (Jyväskylä, Finland, 30 May–3 June 1999), Wiley, New York, 1999, pp. 199–229.

[6] P.J. Bentley, J.P. Wakefield, Generic representation of solid geometry for genetic search, Microcomputers in Civil Engineering 11 (3) (1996) 153–161.

7 J.A. Biles, GenJam: a genetic algorithm for generating jazz solos, International Computer Music Conference (San Francisco, CA), International Computer Music Association, 1994 pp. 131–137.

[8] R.R. Birge, Protein-based optical computing and memories, Computer 25 (11) (1992) 56–67.

[9] C. Caldwell, V.S. Johnston, Tracking a criminal suspect through 'face-space' with a genetic algorithm, Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991 pp. 416–421.

[10] S.C. Cheung, J. Kramer, Compositional reachability analysis of finite-state distributed systems with user-specified constraints, Proceedings of SIGSOFT'95 Third ACM SIGSOFT Symposium on the Foundations of Software Engineering 1995 pp. 140–151.

[11] O. Cordón, F. Herrera, L. Sánchez, Evolutionary learning processes for data analysis in electrical engineering applications, in: D. Quagliarella, J. Périaux, C. Poloni, G. Winter (Eds.), Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, Wiley, Chichester, 1998, pp. 205–224.

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

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

[14] C.-W. Feng, L. Liu, S.A. Burns, Using genetic algorithms to solve construction time-cost trade-off problems, Journal of Computing in Civil Engineering 10 (3) (1999) 184–189.

[15] N.E. Fenton, Software measurement: a necessary scientific basis, IEEE Transactions on Software Engineering 20 (3) (1994) 199–206.

[16] F. Glover, Tabu search: a tutorial, Interfaces 20 (1990) 74–94.

[17] D.E. Goldberg, Genetic Algorithms in Search, Optimization & Machine Learning, Addison-Wesley, Reading, MA, 1989.

[18] S. Helmreich, Recombination, rationality, reductionism and romantic reactions — culture, computers, and the genetic algorithm, Social studies of Science 28 (1) (1998) 39–71.

[19] W.M. Jenkins, The genetic algorithm-or can we improve design by breeding, IEE Colloquium on Artificial Intelligence in Civil Engineering (London, UK, 16 January 1992), IEE, London, 1992 pp. 1/1–4.

[20] B. Jones, H.-H. Sthamer, D. Eyres, Automatic structural testing using genetic algorithms, The Software Engineering Journal 11 (1996) 299–306.

[21] B.F. Jones, D.E. Eyres, H.H. Sthamer, A strategy for using genetic algorithms to automate branch and fault-based testing, The Computer Journal 41 (2) (1998) 98–107.

[22] C.L. Karr, S.K. Sharma, W.J. Hatcher, T.R. Harper, Fuzzy control of an exothermic chemical reaction using genetic algorithms, Engineering Applications of Artificial Intelligence 6 (6) (1993) 575–582.

[23] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.

[24] J.E. Labossiere, N. Turrkan, On the optimization of the tensor polynomial failure theory with a genetic algorithm, Transactions of the Canadian Society for Mechanical Engineering 16 (3–4) (1992) 251–265.

[25] Y. Miyamoto, Y. Miyatake, S. Kurosaka, Y. Mori, A parameter turning for dynamic simulation of power plants using genetic algorithms, Electrical Engineering in Japan 115 (1) (1995) 104–113.

[26] J. Ning, K. Miriyala, W. Kozaczynski, An architecture-driven, business-specific, and component-based approach to software engineering, Proceedings of the Third International Conference on Software Reuse 1994 pp. 84–93.

[27] Y. Ou, On using UML class diagram for object-oriented database design — specification of integrity constraints, in: J. Bézivin, P.-A. Muller (Eds.), The Unified Modeling Language, UML98 — Beyond the Notation, First International Workshop, Mulhouse, France, June 19981998, pp. 147–156.

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

[29] R. Poli, S. Cagnoni, G. Valli, Genetic design of optimum linear and nonlinear QRS detectors, IEEE Transactions on Biomedical Engineering 42 (11) (1995) 1137–1141 November.

[30] R.S. Pressman, Software Engineering: A Practitioner's Approach, 4th ed, McGraw-Hill, New York, NY, 1997 European adaptation. Adapted by Darrel Ince. ISBN 0-07-052182-4.

[31] C. Ryan, P. Walsh, The evolution of provable parallel programs, in: J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo (Eds.), Genetic Programming 1997: Proceedings of the Second Annual Conference, (Stanford University, CA, USA, 13–16 July 1997), Morgan Kaufmann, Los Altos, CA, 1997, pp. 295–302.

[32] E. Sanchez, H. Miyano, J.P. Brachet, Optimization of fuzzy queries with genetic algorithms. Applications to a data base of patents in biomedical engineering, Proceedings of the Sixth International Fuzzy Systems Association World Congress (IFSA'95) 2 (1995) 293–296 Sao Paulo.

[33] M. Sebag, M. Schoenauer, H. Maitournam, Parametric and non-parametric identification of macro-mechanical models, in: D. Quagliarella, J. Périaux, C. Poloni, G. Winter (Eds.), Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, Wiley, Chichester, 1998, pp. 327–340.

[34] M.J. Shepperd, Foundations of Software Measurement, Prentice Hall, Englewood Cliffs, NJ, 1995.

[35] I. Sommerville, Software Engineering, 6th ed, Addison-Wesley, Reading, MA, 2001.

[36] N. Tracey, J. Clark, K. Mander, Automated program flaw finding using simulated annealing, International Symposium on Software Testing and Analysis, ACM/SIGSOFT, 1998 pp. 73–81.

[37] P.J.M. van Laarhoven, E.H.L. Aarts, Simulated Annealing: Theory and Practice, Kluwer, Dordrecht, 1987.

[38] J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, B.F. Jones, Systematic testing of real-time systems, in: Fourth International Conference on Software Testing Analysis and Review (EuroSTAR 96), 1996.

[39] J. Wegener, H. Sthamer, B.F. Jones, D.E. Eyres, Testing real-time systems using genetic algorithms, Software Quality 6 (1997) 127–135.

[40] R. Weiss, J.T.F. Knight, Engineered communications for microbial robotics, Proceedings of the Sixth DIMACS Workshop on DNA Based Computers, University of Leiden, Leiden, The Netherlands, 13–17 June 2000, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, Leiden Center for Natural Computing, Leiden, 2000 pp. 5–19.

[41] D. Whitley, A genetic algorithm tutorial, Statistics and Computing 4 (1994) 65–85.

[42] D. Whitley, A free lunch proof for gray versus binary encodings, in: W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (Orlando, Florida, USA, 13–17 July 1999), vol. 1, Morgan Kaufmann, Los Altos, CA, 1999, pp. 726–733.

[43] D. Whitley, An overview of evolutionary algorithms: practical issues and common pitfalls, Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms, 2001. This volume.

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