ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/228671024

Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications

Article · May 2009

CITATIONS
196

READS
2,562

3 authors:

Mark Harman
University College London
446 PUBLICATIONS **16,130** CITATIONS
SEE PROFILE

Afshin Mansouri
Brunel University London
50 PUBLICATIONS **1,985** CITATIONS
SEE PROFILE

Yuanyuan Zhang
University College London
29 PUBLICATIONS **1,650** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

FITTEST - Future Internet Testing View project

Improving Software Processes via Empirical Game Theory View project

All content following this page was uploaded by Yuanyuan Zhang on 30 May 2014.

The user has requested enhancement of the downloaded file.

Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications

Mark Harman, S. Afshin Mansouri and Yuanyuan Zhang
April 9, 2009

Technical Report TR-09-03

***Abstract*—In the past five years there has been a dramatic increase in work on Search Based Software Engineering (SBSE), an approach to software engineering in which search based optimisation algorithms are used to address problems in Software Engineering. SBSE has been applied to problems throughout the Software Engineering lifecycle, from requirements and project planning to maintenance and re-engineering. The approach is attractive because it offers a suite of adaptive automated and semi-automated solutions in situations typified by large complex problem spaces with multiple competing and conflicting objectives.**

This paper[1] provides a review and classification of literature on SBSE. The paper identifies research trends and relationships between the techniques applied and the applications to which they have been applied and highlights gaps in the literature and avenues for further research.

I. INTRODUCTION

Software Engineering 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 Software Engineering questions.

1) What is the smallest set of test cases that cover all branches in this program?
2) What is the best way to structure the architecture of this system?
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, Software Engineering management and refactoring respectively. It would appear at first sight, that these questions 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 *optimisation* questions. As such, they are typical of the kinds of problem for which Search Based Software Engineering (SBSE) is well adapted and with which each has been successfully formulated as a search based optimisation problem. That is, as we shall see in this survey, SBSE has been applied to testing design, requirements, project management and refactoring. As this survey will show, work on SBSE applied to each of these five areas addresses each of the five questions raised above. The breadth of applicability is one of the enduring appeals of SBSE.

In Search Based Software Engineering, the term 'search' is used to refer to the metaheuristic search-based optimisation techniques that are used. SBSE seeks to reformulate Software Engineering problems as search-based optimisation problems (or 'search problems' for short). This is not to be confused with textual or hypertextual searching. Rather, for Search Based Software Engineering, 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.

This paper aims to provide a comprehensive survey of SBSE. It presents research activity in categories, drawn from the ACM subject categories within Software Engineering. For each, it lists the papers, drawing out common themes, such as the type of search technique used, the fitness definitions and the nature of evaluation.

As will be seen, a wide range of different optimisation and search techniques can and have been used. The most widely used are local search, simulated annealing, genetic algorithms and genetic programming. The paper provides a brief overview of these three most popular search based optimisation techniques in order to make the paper self-contained and to give a comparative flavour for the kinds of search based optimisation that have proved useful in Software Engineering.

As the paper reveals, 59% of the overall SBSE literature are concerned with Software Engineering applications relating to testing. In 2004, McMinn published an extensive survey

Mark Harman and Yuanyuam Zhang are with the Department of Computer Science, King's College London, UK {(Mark.Harman,Yuanyuan.Zhang)@kcl.ac.uk}. Afshin Mansouri is with Brunel Business School, Brunel University, UK {Afshin.Mansouri@brunel.ac.uk}

[1]The paper is a (significantly) extended version of the recent ICSE 'Future of Software Engineering paper' by Harman, one of the present authors [215].

of work on SBSE applications in the general area Software Testing [340], while in 2009 a more recent survey, focusing specifically on testing non functional requirements was presented by Afzal *et al.* [6]. Also recently, a survey of SBSE at the design level was presented by Raiha [385].

Overviews of the more general SBSE area have also been provided. For example, between 2004 and 2006, Mantere and Alander [334], Jiang [255] and Rela [392] provided an overview of Software Engineering by means of evolutionary algorithms, while Clark *et al.* [118] provided an early initial survey and proposal for SBSE development in 2003.

These specific reviews on testing and design aspects of SBSE and the overviews of the area are a testament to the growing importance of and interest in the field of SBSE research. Indeed, as this comprehensive SBSE survey will show, 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 above, there remains, to date, no comprehensive survey of the whole field of study.

The field is now growing so rapidly that this point in time may the last at which it is possible to capture the whole field within a single survey paper. It is, therefore, timely to review the SBSE literature, the relationships between the applications to which it has been applied, the techniques brought to bear, trends and open problems. By seeking to provide a comprehensive survey, we are able to capture the trends and relationships between sub areas of SBSE research and to identify open problems and areas that remain little explored in the literature to date.

The primary contributions of this survey are as follows:

1) **coverage and completeness**: The field is sufficiently mature to warrant a survey paper, but not so large that it becomes impossible to cover all the important papers. This survey seeks to gather together all work in a complete survey covering SBSE from its early origins to a publication 'census date' of December 31st 2008. This census date is chosen for pragmatic reasons. As this survey reveals, there is a notably increasing trend of publication in SBSE. This survey already reports on the work of just over 500 publications. The growth in activity in this area makes a survey useful, but it also means that it may not be feasible to conduct a detailed survey after this date.
2) **classification**: The classification of Software Engineering areas allows us to identify gaps in the literature, indicating possible areas of Software Engineering that could, but have yet to, benefit from the application of SBSE. Similarly, the analysis of search techniques used, allows us to identify search based optimisation algorithms that have yet to receive significant attention. We also apply Formal Concept Analysis (FCA) [439] in order to explore the relationships between techniques and the applications to which they have been applied.
3) **Trend analysis**: The survey presents numeric data concerning trends which give a quantitative assessment of the growth in the area and the distributions of activity among the Software Engineering domains that have received attention. We are also able to identify recent growth areas.

The rest of the paper is organised as follows: Section II provides a brief overview of the search based optimisation algorithms that have been most widely used in SBSE research areas. Section III describes the publications classification scheme used in this paper. Section IV, V, VI, VII, VIII, IX and X present the characteristics of related literature according to the seven main categories in detail, that is: Requirements/Specifications; Design Tools and Techniques; Software/Program Verification and Model Checking; Testing and Debugging; Distribution, Maintenance and Enhancement; Metrics and Management. Section XI explores the relationship between the techniques employed and their applications in Software Engineering. Section XII shows how the general nature of SBSE can provide a bridge between apparently unrelated areas in Software Engineering. Section XIII discusses overlooked and emerging areas in the literature. Section XIV reviews the benefits that can be expected from further development of the field of SBSE. Section XV concludes.

II. Background

Although interest in SBSE has witnessed a recent dramatic rise, its origins can be traced back to early work on optimisation in software engineering in the 1970s. One of the earliest attempts to apply optimisation to a Software Engineering problem was reported by Miller and Spooner [355] in 1976 in the area of software testing. To the best of our knowledge, Xanthakis *et al.* [493] were the first apply a metaheuristic search technique to a software engineering problem in 1992.

The term SBSE was first used by Harman and Jones [218] in 2001. This was the first time in the literature that it was suggested that search based optimisation could be applied right across the spectrum of Software Engineering activity. The position paper acted as a form of 'manifesto' for SBSE. However, it should also be noted that, much earlier, Carl Chang has also used his IEEE Software editorial to promote the more widespread use of Evolutionary Computation in Software Engineering in 1994 [100].

The UK Engineering and Physical Science Research Council (EPSRC) provided funding for a network called SEMINAL: Software Engineering using Metaheuristic INnovative ALgorithms[2]. SEMINAL ran from 1999 to 2002. It was instrumental in the early growth of the SBSE field. SEMINAL held several workshops and events [220] that helped to draw together a community of Software Engineers interested in applying metaheuristic search techniques to Software Engineering problems. The network participants published an initial survey of the area in 2003 [118]. The EPSRC and the EU continue to provide significant funding for the development of SBSE through the large EPSRC project SEBASE (Software Engineering by Automated Search) and the EU Evolutionary Testing (EvoTest) project.

[2]The SEMINAL acronym was suggested work of David Corne.

The number of publications[3] on SBSE shows a dramatically increasing trend since 2001, not merely because of the work of these two large multisite projects, but because of the even larger number of researchers, worldwide, who have found considerable value in the application of search based optimisation to an astonishingly wide variety of applications in Software Engineering. Figure 1 provides a histogram charting this publication growth over time, while Figure 2 shows the proportion of papers that fall into each of the different Software Engineering application area subject categories.

The rest of this section provides a brief overview of the Search Based optimisation algorithms that have been most widely used in SBSE research: global search techniques such as genetic algorithms and simulated annealing and local search techniques such as hill climbing. This overview is sufficient to make the paper self-contained. A more detailed treatment of these algorithms and many others available to practitioners and researchers in SBSE can be found in the recent survey of search methodologies edited by Burke and Kendall [93].

A. Key Ingredients

There are only two key ingredients [215, 218] for the application of search-based optimisation to Software Engineering problems:

1) The choice of the representation of the problem.
2) The definition of the fitness function.

This simplicity and ready applicability makes SBSE a very attractive option. Typically, a software engineer will have a suitable representation for their problem, because one cannot do much engineering without a way to represent the problem in hand. Furthermore, many problems in Software Engineering have a rich and varied set of software metrics associated with them that naturally form good initial candidates for fitness functions [217].

With these two ingredients it becomes possible to implement search based optimisation algorithms. These algorithms use different approaches to locate optimal or near optimal solutions. However, they are all essentially a search through many possible candidate instances of the representation, guided by the fitness function, which allows the algorithm to compare candidate solutions according to their effectiveness at solving the problem in hand (as measured by the fitness function).

1) Hill Climbing: Hill Climbing (HC) starts from a randomly chosen initial candidate solution. At each iteration, the elements of a set of 'near neighbours' to the current solution are considered. Just what constitutes a 'near neighbour' must be defined for each application to which hill climbing is applied, because it is problem specific. Typically, the determination of a mechanism for identifying near neighbours is relatively straightforward; the near neighbours are other candidate solutions that are a 'small mutation away' from the current solution.

[3]We have made available a repository of all papers on SBSE cited in this literature survey. The repository is a searchable resource for the growing SBSE community that contains full details of papers, abstracts and citation information and pointers to the Digital Object Identifier (DOI) for the reference. It can be fount at http://www.sebase.org/sbse/publications/.

At each iteration of the main loop, the hill climbing algorithm considers the set of near neighbours to the current solution. A move to a new current solution is taken if a near neighbour can be found with a better fitness value. There are two choices:

1) In next ascent hill climbing, the move is made to the first neighbour found to have an improved fitness.
2) In steepest ascent hill climbing, the entire neighbourhood set is examined to find the neighbour that gives the greatest increase in fitness.

If there is no fitter neighbour, then the search terminates and a (possibly local) maxima has been found. That is, by definition, since there is no near neighbour with a higher fitness value, we must be at a maxima. However, it may not be a global maxima; there may be other maxima with higher fitness that could have been reached from a different starting point. Figuratively speaking, a 'hill' in the search landscape close to the random starting point has been climbed.

Clearly, the problem with the hill climbing approach is that the hill located by the algorithm may be a local maxima, and may be far poorer than a global maxima in the search space. For some landscapes, this is not a problem, because repeatedly restarting the hill climb at a different location may produce adequate results. This is known as multiple-restart hill climbing and often when hill climbing is used, it is used with multiple restarts. Despite the local maxima problem, hill climbing is a simple technique that is both easy to implement and surprisingly effective in many SBSE problems [224, 359].

2) Simulated Annealing: Simulated Annealing (SA) [348] can be thought of as a variation of hill climbing that avoids the local maxima problem by permitting moves to less fit individuals. The approach gives more chances to consider less fit individuals in the earlier stages of the exploration of the search space. This chance is gradually reduced until the approach becomes a traditional hill climb in the very final stages of the exploration of the search space.

Simulated annealing is a simulation of metallurgical annealing, in which a highly heated metal is allowed to reduce in temperature slowly, thereby increasing its strength. As the temperature decreases, the atoms have less freedom of movement. However, the greater freedom in the earlier (hotter) stages of the process allow the atoms to 'explore' different energy states.

A simulated annealing algorithm will move from some point x_1 to a *worse* point x_1' with a probability that is a function of the drop in fitness and a 'temperature' parameter that (loosely speaking) models the temperature of the metal in metallurgical annealing. The effect of 'cooling' on the simulation of annealing is that the probability of following an unfavourable move is reduced. The earlier 'warmer' stages allow productive exploration of the search space, with the hope that the higher temperature allows the search to escape local maxima. The approach has found application in several problems in SBSE [48, 77, 359, 451]. The algorithm is described in Figure 3.

3) Genetic Algorithms: Genetic Algorithms (GAs) use concepts of population and of recombination [245]. Of all optimisation algorithms, genetic algorithms have been the most widely applied search technique in SBSE, though this may

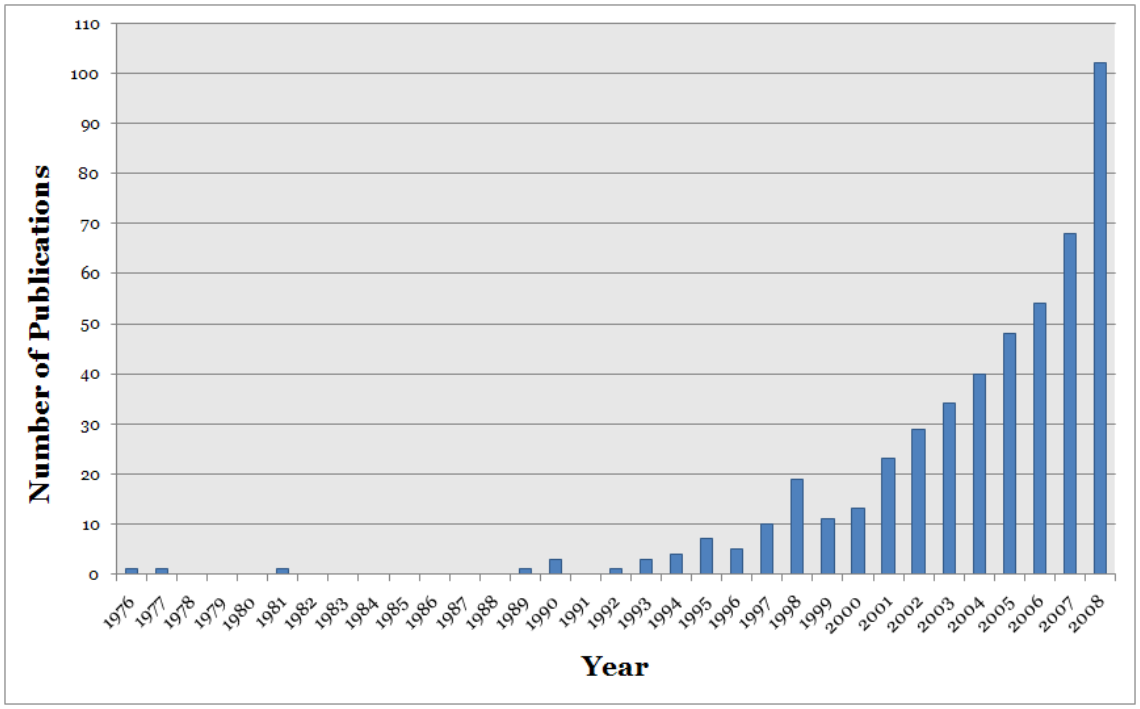


Fig. 1. The trend of publications on SBSE.

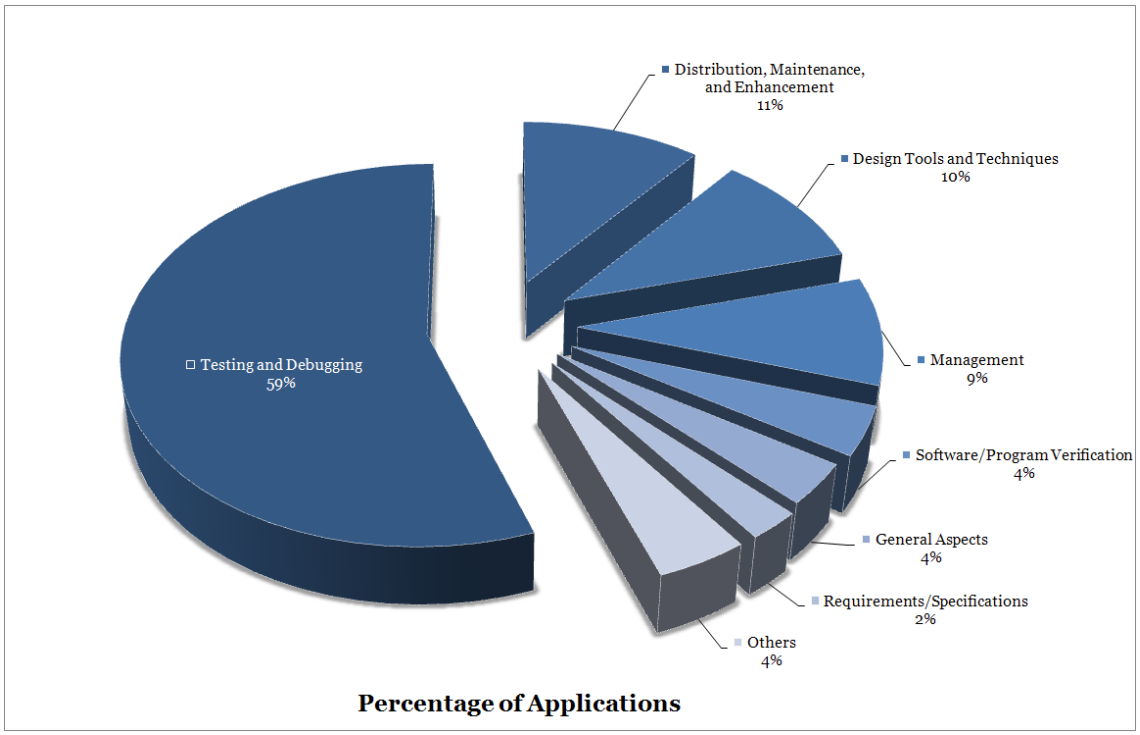


Fig. 2. Spread of SBSE papers reviewed in this article based on their application areas activities

```
Initialise x to x_0 and T to T_0

loop — Cooling
  loop — Local search
    Derive a neighbour, x', of x
    ΔE := E(x') − E(x)
    if ΔE < 0
    then x := x'
    else derive random number r ∈ [0, 1]
      if r < e^(−ΔE/T)
      then x := x'
      end if
    end if
  end loop — Local search
  exit if pre-defined stopping condition is satisfied
  T := C(T)
end loop — Cooling
```

Fig. 3. A Generic Simulated Annealing Algorithm

have largely been for historical or sociological reasons, rather than scientific or engineering reasons. That is, perhaps genetic algorithms, with their scientifically nostalgic throw back to Darwin, simply have more natural appeal to researchers who, presented with a potentially bewildering array of possible search based optimisation techniques, have to choose one with which to begin experimentation on a new Software Engineering field of application.

Certainly, there have been few studies that seek to establish the theoretical or practical performance differences between different search based algorithms for SBSE. This is perhaps a reflection of the youth of the field; the literature is currently witnessing what might be termed the 'gold rush' phase in the development of a new sub-discipline. In such an early gold rush phase, many authors are discovering highly novel ways of using search based optimisation in Software Engineering problem domains for the very first time.

A generic genetic algorithm [118] is presented in Figure 4. An iterative process is executed, initialised by a randomly chosen population. The iterations are called generations and the members of the population are called chromosomes, because of their analogs in natural evolution. The process terminates when a population satisfies some pre-determined condition (or a certain number of generations have been exceeded). On each generation, some members of the population are recombined, crossing over elements of their chromosomes. A fraction of the offspring of this union are mutated and, from the offspring and the original population a selection process is used to determine the new population. Crucially, recombination and selection are guided by the fitness function; fitter chromosomes having a greater chance to be selected and recombined.

There are many variations on this overall process, but the crucial ingredients are the way in which the fitness guides the search and the recombinatory and the population based nature of the process. There is an alternative form of evolutionary algorithms, known as evolution strategies [417], developed independently of work on Genetic Algorithms. However, evolution strategies have not been applied often in work on SBSE. A good example of a topic area for which ES has been applied can be found is the work of Alba and Chicano [23], who show that evolution strategies may outperform genetic algorithms for some test data generation problems.

```
Set generation number, m:= 0
Choose the initial population of candidate solutions, P(0)
Evaluate the fitness for each individual of P(0), F(P_i(0))
loop
  Recombine: P(m) := R(P(m))
  Mutate : P(m) := M(P(m))
  Evaluate: F(P(m))
  Select: P(m + 1) := S(P(m))
  m := m + 1
  exit when goal or stopping condition is satisfied
end loop;
```

Fig. 4. A Generic Genetic Algorithm

There is also a variation of genetic algorithms, called Genetic Programming (GP) [284], in which the chromosome is not a list, but a tree. The tree is the abstract syntax tree of a program that is evolved using a similar genetic model to that employed by a genetic algorithm. Genetic programs are typically imperfect programs that are, nonetheless, sufficiently good for purpose. Fitness is usually measured using a testing-based approach that seeks to find a program best adapted to its specification (expressed as a set of input/output pairs). Genetic programming has been used in SBSE to form formulæ that capture predictive models of software projects [148, 149] and in testing [469, 473], where genetic programming is well adapted to the problem of testing OO code, for which the test inputs can be sequences of method calls.

III. Classification Scheme

In this analysis and survey paper, we employ the classification scheme which is described in Table I. Our classification of Software Engineering activities is taken from the ACM Computing Classification System, projected onto those Software Engineering areas to which SBSE has been applied. The SBSE literature is categorized using the classification criteria in order that the current research focus as well as avenues for further research can be identified.

IV. Requirements/Specifications

Requirements engineering is a vital part of the Software Engineering process [109], to which SBSE has also been applied in order to optimise choices among requirements, the prioritization of requirements and the relationships between requirements and implementations. A summary of the papers addressing search based Requirements/Specifications (ACM: D.2.1) are summarized in Table III.

Bagnall *et al.* [47] formulated the 'Next Release Problem (NRP)' as a search problem. In the NRP, the goal is to find the ideal set of requirements that balance customer requests, resource constraints, and requirement interdependencies.

Classification Criteria	Values
Type of activity (ACM coding)	Network Protocols (C.2.2), Requirements/Specifications (D.2.1), Design Tools and Techniques (D.2.2), Coding Tools and Techniques (D.2.3), Software/Program Verification (D.2.4), Testing and Debugging (D.2.5), Distribution, Maintenance and Enhancement (D.2.7), Metrics (D.2.8), Management (D.2.9), Distributed Artificial Intelligence (I.2.11), Security and Protection (K.6.5)
Objectives (or fitness)	Maximum Cohesion, Minimum Coupling, ...
Representation method	Tree, Graph, String, *etc.*
Search techniques	Greedy Search, Hill Climbing, Genetic Algorithms, Simulated Annealing, Tabu Search, Other Search Techniques
Problems used for evaluation	Real World Data, Synthetic Data

TABLE I
THE CLASSIFICATION SCHEME FOR SBSE LITERATURE.

Feather and Menzies [166] applied Simulated Annealing, building an iterative model to address search based requirements analysis as problems of selection and optimisation. The authors proposed a Defect Detection and Prevention (DDP) process based on a real-world instance; a NASA pilot study. The DDP combined the requirements interaction model with the summarization tool to provide and navigate the near-optimal solutions in the cost-benefit tradeoff space. The paper was one of the first to use pareto optimality in SBSE, though the pareto fronts were not produced using multi objective optimisation techniques (as with more recent work), but were produced using the iterative application of a single objective formulation. Harman *et al.* [229] also consider requirements problems as feature subset selection problems, presenting results on a single objective formulation for a real world data set from Motorola.

Bagnall *et al.* applied a variety of techniques, including greedy algorithms and simulated annealing to a set of synthetic data created to model features for the next release and the relationships between them. The NRP is an example of a feature subset selection search problem. Baker *et al.* [48] address the problem of determining the next set of releases of a software via ranking and selection of candidate software components. They use greedy and simulated annealing algorithms.

Greer and Ruhe proposed a GA-based approach for planning software releases [200]. Like many problems in software engineering, such as project planning, NRP and regression testing, there is a relationship between feature Subset Selection problems and Feature Ordering (Prioritization) problems. A comparison of approaches (both analytical and evolutionary) for prioritizing software requirements is proposed by Karlsson *et al.* [267]. Greer [200] also provides a method for optimally allocating requirements to increments, based on

1) A means of assessing and optimizing the degree to which the ordering conflicts with stakeholder priorities within technical precedence constraints.
2) A means of balancing required and available resources for all increments.
3) An overall method for continuous planning of incremental software development based on a genetic algorithm.

Zhang *et al.* [508] provide a multi-objective formulation of the next release problem (NRP) to optimise value and cost. They present the results of an empirical study into the suitability of multi-objective search techniques.

Early work on integration by Saliu and Ruhe [412] showed how implementation objectives and requirements objectives could be simultaneously optimised using a multi-objective optimisation approach. Like Zhang *et al.* [508], this work also formulates the problem as a two-objective pareto optimal problem, but in this case with implementation level and requirement level objectives, where as Zhang *et al.* use cost and value as their two objectives.

Finkelstein *et al.* [181] showed how a multi-objective optimisation approach can be used to explore fairness of outcomes in requirements assignments. There are different definitions of fairness. For example, each customer might wish to receive equal spend from the developers, or they might prefer that they receive an equal number of their desired requirements compared to other customers. Finkelstein *et al.* show how these different definitions of fairness can be considered to be different objectives to be optimised.

Also, with relevance to multi objective pareto optimal formulations, Feather *et al.* [167, 168] summarize the visualization techniques used to present requirements status, including Pareto fronts plotted by Simulated Annealing. Jalali *et al.* [251] also consider the requirements optimisation problem. They use Greedy Algorithms to reduce risks and increase the number of requirements achieved.

The application of SBSE optimisation techniques to requirements analysis problems provides one example of a Software Engineering application that is often regarded as inherently imprecise, qualitative and informal. However, using SBSE it can be formalized as a quantitive multi-objective optimisation problem. A position paper on recent trends in requirements analysis optimisation is provided by Zheng *et al.* [509].

V. DESIGN TOOLS AND TECHNIQUES

In other engineering disciplines search based optimisation is widely used as a means of developing better designs. Where there are widely accepted metrics, such as cohesion and coupling, there has been much work on optimizing these [152, 224, 228, 323, 326, 327, 359, 360, 362, 363, 364]. However, this previous work on cohesion and coupling, is not concerned with design *per se*. Rather, it is concerned with the problem of re-constructing the module boundaries of a system after implementation. Though it uses design criteria, such as cohesion and coupling, it uses these to re-partition the module boundaries of existing software. As such, this previous work is categorized as work on maintenance, rather than work on design in this survey. Raiha [385] provides a recent detailed survey of SBSE techniques for both design problems and re-design (maintenance) problems in Software Engineering.

This section reports on the existing work on SBSE for Software Engineering design. A summary of the papers addressing activities related to search based Design Tools and

Techniques (D.2.2) are summarized in Table IV. More work is required to assess the impact of design choices on down stream development, testing and maintenance in such a way that design decisions can be measured and assessed for their impact on these downstream activities. When this becomes better understood, it is likely that SBSE will follow with techniques for design optimisation based on the expected downstream benefits that will accrue from the available alternatives for design choices.

Clearly, there is a relationship between re-design (for software maintenance) and design (as a part of the initial design phase of the lifecycle). This relationship is borne out naturally in the literature on software design, where some of the SBSE techniques from software maintenance also have been adapted for software design. Simons and Parmee [434, 435, 436, 437] propose multi-objective GA to address object-oriented software design. Like the previous work on cohesion and coupling for software maintenance [224, 228, 326, 327, 360, 362], the fitness function is inspired by similar Software Engineering goals. However, the goal is upstream software design rather than more downstream maintenance. O'Keeffe and Ó Cinnéide [371, 372] convert object-oriented software design to an optimisation problem using SA. A set of metrics is used for evaluating the design quality. This is a development of work by the same authors on refactoring OO systems according to metrics (which is described in Section VIII-B).

It would be natural to suppose that work on design patterns [185] could and should form a foundation for a strand of work on SBSE for design. This possibility has recently been explored in detail by Raiha *et al.* [383, 384, 386, 387], who propose a GA–based approach to automatically synthesize software architectures consisting of several design patterns.

Other authors have proposed new SBSE approaches, specifically targeted at the design phase of the software development process. Feldt [174] presents a model to explore the difficulty in early software development phases by using GP and also describes a prototype of interactive software development workbench called WISE that uses biomimetic algorithms [175]. Several authors have also considered SBSE techniques for balancing quality of service objectives, such as Khoshgoftaar *et al.* Khoshgoftaar *et al.* [274, 275], who propose an approach for calibrating a multi-objective module-order model (MOM) using GP.

The problem of (Quality of Service (QoS) aware web service composition was introduced by Canfora *et al.* [95], who use Genetic Algorithms to solve QoS-aware composition optimisation problem. This problem, which lies at the heart of Service Oriented Computing, implemented as web–based systems, has recently been taken up and studied by other authors. Jaeger and Mühl [250] discuss the Quality of service-based web services selection problem using GAs. Ma and Zhang [319] propose a GA–based method for web service composition and web service selection which takes account of QoS constraints. Zhang *et al.* [446, 504, 505] apply GAs for web services selection with global QoS constraints.

Several authors have addressed the design problem of component selection and integration. This component selection problem is closely related to the requirement assignment problem. Baker *et al.* [48] present results on greedy optimisation and simulated annealing for component selection, while Yang *et al.* [498] propose an approach for software integration problem by using GA to reduce risk. Classical OR techniques have also been applied to component selection problems: Desnos *et al.* [139] combine backtracking and branch-and-bound techniques for automatic component substitution problem to optimise software reuse and evolution. Other authors have considered the component selection problem as a selection optimisation problem. For example Cortellessa *et al.* [129] presented a framework to support the selection of Code Off The Shelf (COTS) components in order to minimize the system construction cost, while Vijayalakshmi *et al.* [459] propose a GA-based approach to select an optimised combination of components and Kuperberg *et al.* [285] propose a GP-based platform-independent reengineered parametric behaviour models for black-box components performance prediction.

State based models of design are increasingly popular and these present opportunities for SBSE research because of the wealth of research on synthesis of state based models from examples, using optimisation techniques. Goldsby *et al.* [196, 197, 198] present a evolution-based tool for software behavioral model generation to improve the quality of systems. The system, Avida-MDE, generates a set of communicating state-based models of system behaviour using model inference techniques that allow a finite state machine model to be synthesized from cases. A related approach is used by Lucas and Reynolds [317], who present an evolutionary algorithm for learning deterministic finite automaton (DFA) to optimally assign state labels and compare its performance with the Evidence Driven State Merging (EDSM) algorithm.

Feldt, one of the early pioneers of the application of search based optimisation to Software Engineering showed how fault tolerance could be designed into systems using GP to evolve multiple diverse software variants [169, 171, 172]. This is a novel approach to N-version computing, in which highly fault tolerant systems are created several times, in different ways, to increase robustness. The goal was to increase quality since the GP evolved versions would be qualitatively different from any human-generated 'diverse versions'.

In the traditional N-version computing approach, different teams of programmers are deployed to develop the different (and hopefully, therefore, diverse) solutions to the same problem. Of course, the development of different versions of as system in this manner is a highly expensive solution to the problem of robustness and fault tolerance; it can and has only be used in highly safety-critical situations, where the expense might be justified. Though it was not directly the intention of the work, Feldt's work also showed that, by using GP to evolve the required diverse solutions to the same problem, there is the potential to use SBSE techniques to over come the expense that was previously inherent in N-version computing.

Work on SBSE techniques for design has grown in prevalence in the last three years, with many new and interesting Software Engineering deign problems emerging. Amoui *et al.* [29] apply GAs to optimize OO metrics and find the best sequence of system transformations in order to improve the quality of the transformed design, in an approach that shares

some similarities with work on refactoring using search based optimisation to find good sequences of refactoring steps. Barlas and El-Fakih [55] present a GA–based method for mapping client-server problems to optimise the delivery of applications to multiple clients by multiple servers. Bowman *et al.* [78] apply the SPEA2 multi–objective optimisation algorithm to provide decision support system for the Class Responsibility Assignment (CRA) problem. Cao *et al.* [98] and [97] address the cost-driven web service selection problem by using GA. Chardigny *et al.* [107] and [106] propose a search based approach to the extraction of component-based architectures of OO systems. As with other work in this section, this work could be categorized as design or as re-design, highlighting the interplay in Software Engineering between design, maintenance and evolution of software systems. Sharma and Jalote [427] propose a heuristic-based approach for deploying software components that maximises performance.

VI. Software/Program Verification and Model Checking

A summary of the papers addressing problems related to Software/Program Verification (D.2.4) are summarized in Table VI.

Godefroind was the first to apply search based optimisation to explore the state space used in model checking [194]. Where the state space is too large to be fully checked, search based optimisation can be used to identify isomorphic sub graphs and to seek out counter examples. Alba *et al.* [19, 20, 21, 25, 112, 112, 113] also show how Ant Colony Optimisation (ACO) can be used to explore the state space used in model checking to seek counter examples. Mahanti and Banerjee [321] also propose an approach for model checking, using ACO and Particle Swarm Optimisation techniques.

Other authors have also explored the relationship between SBSE and model checking. For instance Johnson [259] used model checking to measure fitness in the evolution of finite state machines, while Katz and Peled [268, 269] provided a model checking based genetic programming approach for verification and synthesis from specification. He presents an approach that combines Hoare–logic–style assertion based specifications and model checking within a Genetic Programming Framework [237].

VII. Testing and Debugging

Software testing is the process used to measure the quality of developed software. Usually, quality is constrained to such topics as correctness, completeness, security, but can also include non-functional requirements, such as those described under the ISO standard ISO 9126[4], including capability, reliability, efficiency, portability, maintainability, compatibility and usability.

Of all the areas of Software Engineering activity to which SBSE techniques have been applied, software testing is both the first area tackled and that which has received the most widespread study. The relative breakdown of the quantity of publications in each area was given in Figure 2. Our survey reveals that 59% of the papers published on SBSE topics are concerned with software testing.

Indeed, the area was already sufficiently developed that it merited its own survey (of search based testing) in 2004 [340] and is now the topic of an annual international workshop on Search Based Software Testing.

The general idea behind all approaches to search based test data generation is that the set of test cases (or more usually, possible inputs to the program) forms a search space and the test adequacy criterion is coded as a fitness function. For example, in order to achieve branch coverage, the fitness function assesses how close a test input comes to executing an uncovered branch; in order to find worst case execution time, fitness is simply the duration of execution for the test case in question.

The first application of optimisation techniques to Software Engineering problems is typically attributed to Miller and Spooner [355], who advocated the use of numerical maximization as a technique for test data generation for floating point computations. The first use of a search based optimisation technique is typically attributed to Xanthakis *et al.* [493] who used GAs to generate test data for structural coverage. Other early advocates of search based techniques for test data generation (and among the first authors to publish on SBSE) were Schoenauer and Xanthakis [415] who focussed on the development of improved techniques for constraint handling in genetic algorithms, using search based test data generation as an example application.

Davies *et al.* [131] were also early pioneers of Search Based Software Testing. They developed a GA-based approach for test case generation for an expert system used as part of a combat pilot's support system.

Ferguson and Korel [177, 178] were the first authors to use local search for the problem of test case generation. They also showed how static analysis techniques and search based approaches could be combined to seek test inputs that traverse a hard-to-cover predicate branch. The static analysis component involved 'chaining' backwards to find new data dependence form so called 'problem nodes' (nodes for which no covering test input has yet been found). Though it can be an effective way to generate test data, the process of chaining can be very expensive, because it entails a potential exponential explosion in the number of paths that need to be considered. However, the use of local search was inexpensive.

Recently, it has come to be realized that the variation on local search, originally proposed by Korel [281], called the 'Alternating Variable Method' (AVM) is highly effective as well as efficient. In a recent and relatively large-scale empirical study, AVM was found to be more efficient than population-based optimisation techniques (as might be expected) but it was also found to be competitive in terms of test effectiveness [221].

A wide variety of testing goals have been attacked using search, including structural testing, functional and non functional testing, safety testing, robustness testing, stress testing, mutation testing, integration testing and exception testing. A summary of the papers addressing Testing and Debugging

[4]http://www.issco.unige.ch/projects/ewg96/node13.html

(ACM: D.2.5) are summarized in Table VII. The rest of this section considers each area as a subsection. These subsections consider, in more detail, those subareas of testing that have received most attention in the literature.

In addition to these, other less thoroughly explored areas include Functional Testing [91, 479], Safety Testing [53, 161], Security Testing [133], Robustness Testing [416], Stress Testing [80, 132], Integration Testing [79] and Quality of Service Testing [143].

It is interesting to note that many of these areas have only arisen as topic within Search Based Testing in the past few years, suggesting that the field continues to increase in breadth, covering an increasing number of topics within testing.

Program and system level transformation has also been applied to improve the performance and applicability of Search Based Software Testing in what has come to be known as 'Testability Transformation'. Testability Transformation is used to refer to any technique for transforming systems to improve their testability. It has been largely applied to Search Based Testing [54, 225, 226, 227, 242, 345, 347], though it has also been proposed as a solution to other problems in test data generation, beyond search based approaches [216].

Most work on SBSE applied to testing has been concerned with the discovery of faults in software. However, more recently, authors have also turned their attention to the use of SBSE to patch software to fix bugs [36, 41] using co-evolution. Co-evolution has also been used to co-evolve programs and test data from specifications [38]. Patton *et al.* [378] describe a GA approach for software usage testing. The GA generates additional test cases around those that cause a fault in order to help guide a debugger.

Most of the work on Software Testing has concerned the problem of generating inputs that provide a test suite that meets a test adequacy criterion. The schematic overview of all such approaches is presented in Figure 5. Often this problem of test input generation is called 'Automated Test Data Generation (ATDG)' though, strictly speaking, without an oracle, only the input is generated. Figure 5 illustrates the generic form of the most common approach in the literature, in which test inputs are generated according to a test adequacy criteria. The test adequacy criteria is the human input to the process. It determines the goal of testing.

The adequacy criteria can be almost any form of testing goal that can be defined and assessed numerically. For instance, it can be structural (cover branches, paths, statements) functional (cover scenarios), temporal (find worst/best case execution times) etc. This generic nature of Search Based Testing (SBT) has been a considerable advantage and has been one of the reasons why many authors have been able to adapt the SBT approach different formulations.

The adequacy criteria must be captured by a fitness function. This has to be designed by a human, but once a fitness function has been defined for a test adequacy criterion, C, then the generation of C-adequate test inputs can be automated using SBSE. The SBSE tools that implement different forms of testing all follow the broad structure outlined in Figure 5. They code the adequacy as a fitness, using it to assess the fitness of candidate test inputs. In order to assess fitness, the ATDG system has to cause the program to be executed for the candidate inputs. The ATDG system then monitors the execution to assess fitness (how well does the input meet the test adequacy criterion?).

A. Structural Testing

The most widely studied area of testing research that has been addressed using SBSE is structural testing. This was also the first topic to be addressed using metaheuristic search (by Xanthakis *et al.* [493]). The idea is to instrument the program to measure coverage of some structural criterion. The most commonly considered criteria is branch coverage, though other structural criteria have been attacked. For example, Girgis [193] targets data flow coverage while Xiao *et al.* [495] target decision-coverage.

Early work considered the goal to be the coverage of as many branches as possible, so the representation was a test suite and the fitness function sought to Maximise coverage. However, this was found to produce solutions that achieve reasonable coverage, but not full coverage because they tended to avoid the branches that were hard to cover.

Later work has tended to adopt the approach in which the coverage of each branch is viewed as a test objective in its own right, for which a fitness function is constructed based upon the path taken by a test case. The representation is an individual test case.

More recently, Lakhotia *et al.* [230] introduced a multi-objective paradigm for structural test data generation that seeks to Maximise coverage while also achieving other objectives. This is the first work to combine non-functional testing goals with coverage based adequacy criterion as a multi objective problem.

From 1995 there has been an upsurge in interest in Search Based test data generation for structural criteria, based on the achievement of branch coverage [9, 26, 27, 30, 70, 86, 87, 108, 158, 179, 192, 240, 260, 261, 263, 271, 286, 289, 297, 306, 307, 311, 312, 339, 344, 349, 350, 351, 352, 353, 354, 356, 377, 380, 382, 388, 399, 400, 401, 440, 441, 442, 444, 445, 463, 465, 466, 475, 476, 496, 497, 506].

This work focussed on the application of evolutionary algorithms for structural testing, forming a sub area of activity which has come to be known as 'evolutionary testing'. The work culminated in a state-of-the-art evolutionary testing system, used by DaimlerChrysler for automative embedded software assurance [484]. This work is covered in more detail in the survey by McMinn [340]. The Evolutionary Testing approach has also been implemented as a tool by the EU EvoTest project.

Early work on evolutionary testing considered imperative programming paradigms, with much work on structural test data generation for the C programming language. More recently, researchers have adapted these techniques for the OO paradigm [37, 39, 40, 42, 110, 410, 469, 470].

Tonella [450] was the first to apply Search Based testing techniques to the problem of testing Object Oriented (OO) software. He provided a GA-based test case generation technique for unit testing of classes to maximise given coverage

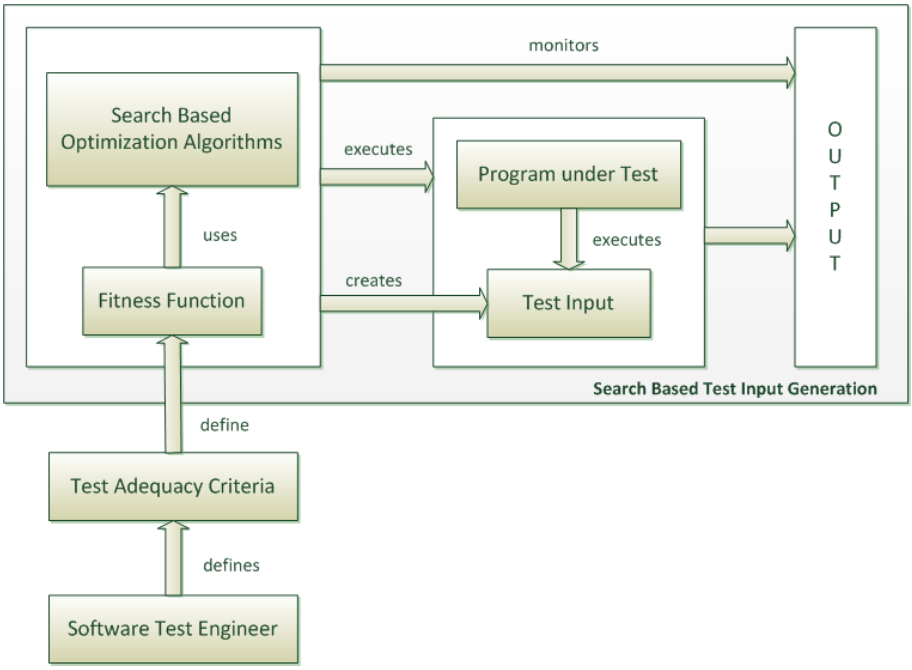


Fig. 5. A Generic Search Based Test Input Generation Scheme

and minimise test set. For the OO paradigm, the representation of a test case needs to be changed, because test cases are typically sequences of method invocations (and their parameters). This makes the testing problem much harder than that typically addressed for the imperative paradigm, in which the problem has been so-defined that the inputs are simply vectors containing numeric input. For the OO paradigm the input has a program-like structure, within which near neighbours are harder to determine and for which small changes in the input can lead to dramatic changes in behaviour.

Since the input to an OO program has a program like structure itself, this naturally suggest that application of genetic programming, which treats the individual to be optimised as a program. This approach has recently been explored by Arcuri and Yao [38]. Though this GP approach is also possible in the imperative paradigm, it has not been explored in the published formulations of the problem.

The OO paradigm also brings with it the state problem, first identified and addressed within the imperative paradigm by McMinn [341, 342, 343]. The state problem occurs when the target branch to be covered is controlled by a predicate, the value of which depends upon the value of some state variable. A state variable is a variable whose value is persistent between different invocations of the function under test. In the imperative paradigm, this occurs through static and global variables. The problem is that the target branch may not be covered by an arbitrary call to the function under test. Rather, the presence of state variables may mean that the target branch must be executed multiple times in order to ensure that the state variable achieves some value.

There has been much other work on structural test data generation for the OO paradigm. Wappler and Wegener [472] were the first to propose the use of Strongly-Typed Genetic Programming for OO test input generation, an approach subsequently adopted by other researchers. [418, 419, 420, 421] and Ribeiro *et al.* [394, 395, 396] use a GP approach for automated test generation for object-oriented software. Ribeiro *et al.* [393, 397, 398] also apply GP to reduce the size of the search space for test case generation in object-oriented software. Arcuri and Yao [40] use and compare the performance of Genetic Algorithms, Hill Climbing and Memetic.

Algorithms for OO test data generation. Liaskos *et al.* [303, 305] use Artificial Immune Systems (AIS) for Definition-Use (DU) path testing of OO software. Arcuri and Yao [42] Compare the performance of four search techniques (HC, SA, GA, MA) and Random Search techniques to generate unit test data for container classes, in order to maximise program coverage and minimise the length of test sequences. Gupta and Rohil [210] present a GA-based test case generation technique for classes in object-oriented software.

Recently, Harman *et al.* [232] showed how search based testing can be adapted to apply to the Aspect Oriented Paradigm, providing the first search based approach to automated test input generation for AOP programs written in AspectJ.

The most popular search technique applied to structural testing problems has been the Genetic Algorithm. However, other authors have experimented with other search based optimisers such as parallel EAs [23], evolution strategies [18], Estimation of Distribution Algorithms (EDAs) [403, 404, 405, 406, 407, 408, 410], Scatter Search [28, 67, 403, 406], Particle Swarm Optimisation (PSO) [293, 492], Tabu Search [145, 146] and Local search [282, 283].

The fitness function is vital to the success of the search in every approach to SBSE. In work on structural testing, several authors have proposed refinements to the fitness function. For example, Baresel and Sthamer [50], Bottaci [72] and Liu *et al.* [309] develop an Evolutionary Testing (ET) System

for automatic testing in presence of flag conditions, adapting fitness to cater for flags, while Bottaci [73] reviews fitness functions used in search based testing proposing alternatives aimed at partially ameliorating problems with flags. Uyar *et al.* [457] propose a fitness evaluation method based on pairwise sequence comparison used in bioinformatics for evolutionary structural testing. The flag problem has been studied widely in its own right, and was the initial motivation for work on Testability Transformation.

Flag variables pose a problem for attempts to apply search based techniques to structural testing. Flag variables are either true or false, and when used in a predicate, they provide little or no guidance to the search (using standard fitness functions). Wappler *et al.* [474] apply a GA-based code transformation approach to address the flag problem in software testing. The problem was first formulated as a testability transformation problem by Harman *et al.* [226, 227] and also studied by Bottaci [72]. Since then it has been studied by several other authors, each of whom has proposed a solution [27, 50, 54, 309, 474].

Research on Search Based Test Data Generation for structural criteria is now relatively mature, with work on predictive methods for test effort [288], theoretical analysis of the problem characteristics [44] and specially tailored search algorithms that are adapted specifically to the structural test data generation problem [346]. Harman and McMinn provide a large scale empirical study to compare Hill Climbing and Genetic Algorithms for structural test data generation [221]. They analyze the theoretical cases in which the Royal Road Theory predicts that GAs will perform well. The empirical results back up the theoretical predictions.

Harman *et al.* [231] showed how traditional Software Engineering dependence analysis techniques can be used to analyze dependence for predicates under test in structural testing. The dependence analysis can be used to reduce the size of the search space for test data generation, thereby improving the efficiency of test data generation. This approach has great promise as a mechanism for combining static and dynamic analysis to reduce search base size. It was applied to branch coverage, but could also be applied to any white box ATDG approach.

B. Model Based Testing

Testing programs specified by models has become increasingly widely studied in the literature on software testing and search based approaches have also witnessed a recent upsurge in activity, largely focussed on techniques for generating test data for state based models. For these models, one frequently studied problem is the construction of Unique Input Output (UIO) Sequences. That is, a sequence of input values, i is a UIO for a state s if the output that occurs when a machine in state s receives i is unique to i.

This allows a tester to know, through black box testing, that the system must have been in state s. UIOs are important in forming testing sequences capable of detecting transition faults (faults where the correct output occurs, but for which the machine arrives at the wrong state). The UIO is used to determine whether the state resulting from a transition is the correct state.

The problem of finding UIOs that are effective and efficient is a natural optimisation problem that has received much recent attention, both in terms of practical investigation of algorithms for optimizing the trade-offs [136, 137, 138, 206, 207, 208, 241] and also in terms of theoretical analysis and characterization of the underlying optimisation problems [295].

Li *et al.* [299, 300] applied an ACO-based approach for automatic test sequence generation in the state-based software testing of UML Statechart diagrams for coverage. The use of ACo targets the problem of shortening the length of the generated test sequences, a general problem for which ACO is known to be very well adapted, but which has not been widely addressed in the literature of test input generation. Guo *et al.* [209] propose heuristics to optimise the fault isolation and identification process in Finite State Machine testing. Lefticaru and Ipate [292] also apply GAs to state diagrams for automatic test data generation for state-based testing. Recent work has also considered test data generation for state based models expressed in the MATLAB simulink framework using traditional models of coverage and mutation testing [500, 501, 502, 503]. Search Based Mutation Testing itself, is covered in more detail in the next section.

C. Mutation Testing

In mutation testing a set of variants of the original program are created. Each variant, called a mutant, is created by inserting a single fault into the original program. A test input is said to kill a mutant if it distinguishes the behaviour of the mutant from that of the original program. That is, the input gives rise to as different output when applied to the mutant and original program. The quality of a test suite can be assessed by measuring the number of mutants killed by the test suite. The more mutants killed, the better the test suite. The idea is that if a test suite is good at killing (detecting) these artificially insert faults, then it will be good at detecting real faults too.

Originally, mutation testing was used as a mechanism for assessing the quality of test suite. However, it was not long [278] before researchers started to consider the problem of generating test data that could be used to kill mutants. In this way, mutation testing became a mechanism for generating test data, rather than merely assessing its quality.

Several authors have presented search-based approaches to the generation of test data to kill mutants. Emer and Vergilio [157] use Genetic Programming to generate and evaluate test cases for the mutation testing. Shan and Zhu [424] use data state mutation for test data generation. In data state mutation, the state of computation is mutated, rather than the program under test. Ayari *et al.* [46] propose Ant Colony Optimisation (ACO) to generate test input data in the automatic mutation testing.

Adamopoulos *et al.* [1] apply a GA for co-evolution of mutants and test cases. This is the first application of co-evolutionary search in the SBSE literature. The idea is to generate sets of mutants and sets of test data that can kill

the mutants. Masud *et al.* [335] also apply GA for test data generation in mutation testing to kill mutants.

Other authors have considered the application of techniques closely related to evolutionary approaches. Baudry *et al.* [58, 59, 60, 61] use Bacteriological Algorithms (BAs) for test data generation for mutation-based testing. BAs are a variant of GAs which retain a population, but remove the cross over operator. May *et al.* [336] apply Artificial Immune Systems (AIS) to mutation testing. AIS is a technique, also inspired by biology, drawing its inspiration from the behaviour of immune system responses. AIS is also used by Liaskos *et al.* [303, 305], though for Definition-Use (DU) path testing of OO software (not mutation testing). May *et al.* [337] describe Artificial Immune System-based test data evolution approach to compare with GA in mutation testing.

Jia and Harman [253, 254] present a higher order approach to mutation testing, in which several faults are simultaneously inserted. There are exponentially more higher order mutants than first order mutants, but Jia and Harman argue that this explosion in size can become manageable using SBSE to search for high quality mutants; those that subsume their first order constituents.

D. Temporal Testing

A lot of the work on search based testing has considered structural and functional properties of the software under test, However, search based optimisation can also be applied to testing non-functional properties, so long as these can be captured by a fitness function. One of the the most conceptually simple applications of the approach has been in temporal testing, where the goal is to search for worst (respectively best) case execution time. In this application the fitness function for a given input i is simply the amount of time taken to execute the program under test for input i. For best case execution time, the objective is to minimise this value, while for worst case execution time the goal is to maximise it. This work has been developed since the mid 1990s, and continues to motivate new research [13, 14, 15, 16, 201, 449, 482, 483].

Of course, worst case execution time can also be approximated using static analysis. However, such an analysis is forced to produce a conservative upper bound on worst case execution time, because the underlying question is undecidable. Dynamic techniques, such as search based testing, can be used to complement approximate static techniques. Wegener and Mueller [481] compare static analysis with ET for the verification of timing constraints.

Where the static and dynamic approaches produce the same answer, we know we have identified the true worst case. In cases where they do not, the gap between the statically and dynamically determined values gives measure of the uncertainly (and thereby the risk) involved. Gross [202, 204] also presented results for temporal testing using evolutionary algorithms, and showed [203, 205] how it was possible to predict the accuracy of these approaches, thereby reducing the uncertainty involved when static and dynamic techniques disagree.

The applications of worst case execution time testing, impinge on highly safety critical aspects of software systems. For instance, Daimler [381, 480, 481] carried out work on the worst case execution time of air bag controllers in production code for embedded systems. It is a sign of the success of the SBSE approach, that it has been used by industrialists such as Daimler for such a safety critical application.

All of the approaches considered so far in this section have used evolutionary computation as the primary search based algorithm for addressing temporal testing. Wegener *et al.* [381, 480, 481] argue that the search space is naturally suited to genetic algorithms. They use three dimensional 'slices' of the N dimensional solution space to show that it is both multi modal and discontinuous and that it contains many plateaux. Plateaux are known to raise issues for optimisation, though there has been recent work addressing this within the optimisation community [489].

This suggests that hill climbing would perform poorly on these problems. However, Dillon [147] presents results which indicate a potential for the use of hybrid approaches incorporating local search (hill climbing) for the determination of worst case execution time for embedded systems.

E. Exception Testing

Exceptions are, by their very nature, hard to test. That is, the conditions for which an exception handler checks are not likely to occur frequently; they are exceptional. Therefore, finding inputs that cause the exception to be raised may not be easy. Indeed, in some systems, there is no input that can raise the exception though, of course, this cannot always be known statically, since it is not a decidable property of a program.

Despite the difficulty of exception testing, there has been work on search based approaches for the problem. At first sight, an exception would appear to denote a 'needle in the haystack' search problem for which search based approaches are known to perform badly (degenerating to random search in the worst case). This is because either a candidate input raises the exception (the desired value from a testing point of view) or it fails to do so (the majority of cases). In order to successfully apply a search based approach, there needs to be a guidance from the non-exception-raising cases to the exception-raising cases.

Fortunately, such guidance is typically available, because the exception raiser itself is typically guarded by a predicate expression that denotes the circumstances under which the exception is to be raised. This makes the problem of exception raising similar to that of branch coverage. Tracey *et al.* [451, 452] were the first to consider this problem. They applied simulated annealing to the problem of raising exceptions (and also to the problems of temporal testing, considered in Section VII-D and structural testing, considered in Section VII-A). This work was later extended to also include genetic algorithms as well as simulated annealing [453, 454, 455]. Boström and Björkqvist [71] use Sequential Quadric Programming (SQP) for the assertion checking problem in Simulink models to automatically generate test cases.

F. Regression Testing

Many organizations have large pools of test data for the purpose of regression testing. Their problem may be more one

of reducing the effort required for regression testing, rather than generating yet more potential test cases for regression testing. Furthermore, if they can Maximise the value and impact of the existing test suite by reducing its size, focussing on those cases with the biggest value, then this may provide additional time and budgetary space for additional test cases that are able to add value.

In these situations, the organization requires a technique for intelligent selection of test cases (test case selection) and for removing any test cases from their test set that can be shown to be subsumed by others (test suite minimization). In the worst case, if the organization runs out of time for regression testing then they need to be sure that they have achieved the best possible results achievable with the test cases that have been used up to the point at which time ran out.

For this regression testing optimisation problem the organization requires a technique for test suite prioritization. That is, to find an ordering of test cases that ensures that, should testing be terminated after some arbitrary prefix of the order has been applied, then the maximum value will have been achieved.

These three problems of selection, minimization and prioritization can all be formulated as optimisation problems to which search based optimisation can be applied. Traditionally, greedy algorithms have been used for minimization and prioritization and a range of techniques involving static analysis have been applied to all three problems [49, 302, 329, 331, 499].

Fischer *et al.* [182, 183] were among the first authors to apply optimisation techniques to software Engineering problems. They used a variant of classical Integer Programming to address the test case selection problem. Their criteria included coverage, like subsequent work on test data generation for testing and test case selection for regression testing. however, they also accounted for connectivity, reachability and date dependence.

Hartmann *et al.* [234, 235, 236] overview and extend strategies for what they term 'automatic selective revalidation', surveying the earlier work of Fischer *et al.* and proposing modifications. The term 'automatic selective revalidation' corresponds roughly to the problem of test case selection in regression testing using the current nomenclature of regression testing.

All three regression testing optimisation problems have also been attacked using search based optimisation. In this work the value of a test suite is typically measured in terms of the coverage it achieves (both in terms of covering changed functionality (for selection), in terms of structural coverage (for all three problems) and in terms of historical fault coverage (for minimization and selection). The cost, against which this value must be balanced by the optimisation technique, is typically measured in terms of the execution time and/or the number of test cases in the test suite.

Li *et al.* [302] use HC and GA for regression test case prioritization. Yoo and Harman [499] introduce a Pareto optimisation approach to the test case selection to optimise: code coverage, time and attaining full coverage with empirical studies of two and three objective formulations using GA and greedy search. This was the first use of pareto optimal multi objective search in regression testing. Bryce and Colbourn [83] propose a hybrid Greedy Algorithm to support test suite selection for achieving coverage. Walcott *et al.* [464] also consider the multi objective problem of balancing coverage with time taken to achieve it. They treat the problem as a single objective problem in which the two objectives are combined into a primary and a secondary objective using weighting.

Several authors [49, 302, 329, 331] have provided comparative studies of the performance of several search based regression testing techniques.

G. Configuration and Interaction Testing

In configuration and interaction testing, the goal is to search the space of possible software configurations for those that are likely to reveal faults. Since configuration spaces can be very large, researchers often focus on levels of coverage of possible interactions between parameter settings for configurations.

The problem was first addressed using search based techniques by Cohen *et al.* [121, 122] and (shortly after) by Ghaze and Ahmed[191]. Cohen has remained active in this field [85, 123, 124]. The problem of interaction testing has also been considered by other authors, who have explored the use of search based solutions [82, 83, 84].

H. Stress Testing

Stress testing consists of finding test cases or scenarios in which software performance degrades. There has been comparatively little work on search based approaches to stress testing. Indeed, non functional test criteria, in general, lag some way behind functional and structural criteria in their development of search problems.

The idea of search based stress testing was mentioned by Alander *et al.* [14]. Mantere [332] investigated GA-based test data generation in system and module testing in a doctoral thesis that considered traditional testing and testing for stressing the SUT and finding bottlenecks. Mantere also suggested the use of Co-evolution for co-evolving GA parameters as a form of co-evolutionary parameter tuning.

Briand *et al.* [80, 81] applied search based optimisation to stress testing problems in which tasks must be completed within a given time interval. A GA was used to identify test scenarios in which deadline misses become more likely. Garousi [186] developed the idea of search based stress testing in a PhD thesis that considered stress testing for distributed real time systems and developed this work with Briand and Labiche [187, 189]. Joshi *et al.* [264] also considered stress testing from the point of view of non functional properties such as power and temperature.

I. Integration testing

Though very important, integration testing has been largely overlooked by the SBSE community, with only a few papers addressing this area of activity. Given the importance of integration testing to the software testing industry and the natural optimisation problems that arise from consideration of

test orders, constraints and selection, this seems like a natural area for further exploration.

If integration order is suboptimal, then there will be a need for an unnecessarily large amount of stubbing to simulate the behaviour of as-yet-unwritten procedures. Hanh *et al.* [212] compare of integration testing strategies including a GA to minimise the stubs and optimise testing resources allocation. Briand *et al.* [79] also address this issue. They use a GA to optimise the orders of test cases in order to minimise the complexity of stubbing in integration testing.

J. Other Testing-Based Applications

Testing is by far the most widely considered area of application for SBSE research. This subsection briefly mentions other work on SBSE applications in testing and debugging that does not fit into any of the previously discussed categories.

Berndt and Watkins [63] search for long test input sequences that simulate extended execution and may be equivalent to the re-execution of a simpler test many times. At the other end of the spectrum, Liu *et al.* [310] seek to generate shortest possible sequences, thereby reducing test effort.

Watkins *et al.* [477] provide GA-based test case generation technique to detect failures in system level testing. They use GA generated test cases to train a decision tree classification system to classify errors. Del Grosso *et al.* [133] use evolutionary testing and dependence analysis to detect buffer overflows. Xiao *et al.* [494] apply EA to generate shortest possible test sequences in protocol conformance testing. Lam *et al.* [287] formulates FSm testing in terms of the asymmetric travelling salesman problem to which ACO can be applied, thereby focussing on the objective of minimizing test sequence length. Briand *et al.* [81] propose GA to improve schedulability of test cases in a real-time system for executing triggered tasks within time constraints. Regehr [391] apply a GA to improve the quality of interrupt scheduling for interrupt-driven software testing. MacNish [320] use a GA to generate test sequences for detecting errors in student code. Chan *et al.* [99] use a GA to detect and identify unwanted behavior of action sequences in computer games. Bueno and Jino [88] provide test case selection and coverage identification metrics based on a GA. Wegener *et al.* [486] use GA-based approach for test case design to improve quality of evolutionary testing in embedded systems. Berndt *et al.* [64] provide GA-based test case generation techniques to improve the quality of test cases. Berndt and Watkins [62] apply GA to automatically generate large volumes of test cases in complex systems. Hermadi [239] briefly survey and compare existing GA-based test data generators. Mayer [338] shows that a reference method in the field of Adaptive Random Testing is not effective for higher dimensional input domains and clustered failure-causing inputs. Feldt [170, 173] uses GP to minimise the probability of coincident failures for multi-version system. Cornford *et al.* [126, 127] extend the Defect Detection and Prevention (DDP) process by using GA and SA to reduce risk in spacecraft systems.

Khurshid [277] proposes and evaluates GA-based techniques to generate test suites for maximizing code coverage and applies it to an implementation of the intentional naming scheme, finding previously undiscovered errors. In order to apply GAs to this problem, Khurshid has to model dynamic data structures, a problem not previously considered.

VIII. Distribution, Maintenance and Enhancement

Software *maintenance* is the process of enhancing and optimizing deployed software (software release), as well as remedying defects. It involves changes to the software in order to correct defects and deficiencies found during field usage as well as the addition of new functionality to improve the software's usability and applicability. A summary of the papers addressing problems in the areas of Distribution, Maintenance and Enhancement (ACM: D.2.7) are summarized in Table VIII.

Much of the work on the application of SBSE to these topics has tended to focus on two strands of research, each of which has attracted a great deal of interest and around which a body of work has been produced.

The first topic to be addressed was Search Based Software Modularization. More recently, there have also been several developments in search based approaches to the automation of refactoring. The previous work on Distribution, Maintenance and Enhancement is discussed in more detail in the following two subsections, which consider work on Modularization and Refactoring separately.

Other work on SBSE application in Distribution, Maintenance and Enhancement that does not fall into these two categories of focussed interest has considered the evolution of programming languages [458], realtime task allocation[5] [57, 156], quality prediction based on the classification of metrics by a GA [461] and legacy systems migration [411]. SBSE has also been applied to the concept assignment problem. Gold *et al.* [195] apply GAs and HCs to find overlapping concept assignments. Traditional techniques (which do not use SBSE) cannot handle overlapping concept boundaries, because the space of possible assignments grows too rapidly. The formulation of this problem as an SBSE problem allows this large space to be tamed.

A. Modularization

Mancoridis *et al.* were the first to address the problem of software modularization using SBSE [326] in 1998. Their initial work on hill climbing for clustering modules to Maximise cohesion and Minimise coupling was developed over the period from 1998 to 2008 [152, 327, 359, 360, 362, 363, 364]. The pioneering work of Macoridis *et al.* led to the development of a tool called Bunch [327] that implements software module clustering and is available for free for research use.

The problem of module clustering is similar to the problem of finding near cliques in a graph, the nodes of which denote modules and the edges of which denote dependence between modules. Mancoridis *et al.* [327] call this graph a Module

[5]This work could equally well be categorised as 'Real Time SBSE'; a topic area which is sure to develop in future, given the highly constrained nature of the real time environment and the many competing objectives that have to be optimised.

Dependency Graph. The Bunch tool produces a hierarchical clustering of the graph, allowing the user to select the granularity of cluster size that best suits their application.

Following Macoridis *et al.*, other authors also developed the idea of module clustering as a problem within the domain of SBSE. Harman *et al.* [224], studied the effect of assigning a particular modularization granularity as part of the fitness function, while Mahdavi *et al.* [322, 323] showed that combining the results from multiple hill climbs can improve on the results for simple hill climbing and genetic algorithms. Harman *et al.* also [228] explored the robustness of the MQ fitness function in comparison with an alternative measure of cohesion and coupling, EVM, used in work on clustering gene expression data.

Other authors have also considered search based clustering problems. Bodhuin *et al.* [68] apply GAs to group together class clusters in order to reduce packaging size and the average downloading times. Huynh and Cai [249] apply GAs to cluster Design Structure Matrices and check the consistency between design and source code structures.

Despite several attempts to improve on the basic hill climbing approach [224, 323, 359], this simple search technique has hitherto been found to be the most effective for this problem. Mitchell and Mancoridis recently published a survey of the Bunch project and related work [361].

Clustering is a very general problem to which a number of algorithms have been applied, not merely search based algorithms. Clustering is likely to find further applications in Software Engineering applications, beyond the original work on software modular structure. For example, Cohen [120] showed how search based clustering algorithms could be applied to the problem of heap allocation Java program optimisation. Memory fragmentation issues have also been addressed using SBSE: Del Rosso [135] improved internal memory fragmentation by finding the optimal configuration of a segregated free lists data structure using GA.

B. Refactoring

In refactoring work, the goal is to change the program, altering its structure without altering the semantics. Closely related topics have also been addressed. For example Reformat *et al.* [389, 390] explore applications of software clones and present a method for automatic clone generation using GP. Clones are also a focus for attention in the work of Di Penta *et al.* [140, 142], who propose a language-independent Software Renovation Framework to remove unused objects, clones and circular dependencies; cluster large libraries into more cohesive and smaller ones. Cowan *et al.* [130] provide a framework of automatic programming applying GP. Bouktif *et al.* [76] use SBSE techniques to schedule refactoring actions in order to remove duplicated code. Antoniol *et al.* [32] propose GA-based refactoring process to reduce size and minimise coupling of libraries. Bodhuin *et al.* [69] introduce a tool to support refactoring decisions using a GA guided by software metrics, in a similar manner to work on refectoring, guided by metrics.

Search Based Refactoring work can be partitioned according to whether the goal is to optimise the program to a refactored version of itself [125, 373, 374, 375, 376, 402, 490] or whether it is to optimise the sequence of refactoring steps to be applied [222, 490]. The work can also be categorized according whether the approach followed is single objective (combining all metrics into a single fitness value) [373, 374, 375, 376, 402, 422, 423, 490] multi objective (using Pareto optimality to separately optimise each metric) [222]. Bouktif *et al.* [76] proposed an approach to schedule refactoring actions under constraints and priorities in order to remove duplicated code.

This work is closely related to that on statement-level Search-Based Transformation, which was first explored by Ryan and Williams in the context of identification of transformations that improve imperative language paralellizability [402, 490] and by Nisbet [370], who apply a GA to determine the optimal transformation sequence that Minimises the execution time of FORTRAN programs for single program multiple data (SPMD) execution on parallel architectures.

Hoste and Eeckhout [246] use multi-objective optimisation to search the space of compiler options that control optimisation levels in `gcc`. There are about 60 such flags (purely for optimisation behaviour of the `gcc`), making for a non-trivial search space of options, specifically targeted at performance of the compiled code. The two objectives considered in the paper are compilation time and code quality (in terms of execution time), though many other possibilities suggest themselves, such as the many non-functional properties of the program being compiled.

Refactoring seeks to restructure a program to improve some aspect of the structure without affecting the behaviour of the restructured system. It is an example of a more general approach: (source-to-source) program transformation, to which SBSE techniques have also been applied. Fatiregun *et al.* [163, 164, 165] and Kessentini *et al.* [270] to apply to transformations to reduce programs size and to automatically construct amorphous slices. The first author considered any form of source-to-source transformation using a search based approach was Cooper [125] who applied search based transformation to find sequences of compiler optimisations. This work used only whole program transformations. The work of Ryan, Williams and Fatiregun which followed, focused on the more 'micro level' or statement-level transformations.

By contrast with this previous work on transformation, the work on refactoring is more concerned with the OO paradigm, but the principles used in the refactoring work are largely the same as those that pertain to the statement-level transformation domain.

IX. Metrics

The fitness function resides at the heart of any approach to search based optimisation. Work on SBSE has a rich vein of research on Software Engineering measurement upon which to draw in order to find fitness functions. It has been argued that all software metrics can be regarded as fitness functions [217]. This allows metrics to move from being merely a passive assessment of a system, process or product, to a more active driver for change. As a fitness function, a metric acquires a new life; it can be used to drive the search based optimisation

process to find better solutions than those currently available. A summary of the papers addressing problems related to Metrics (ACM: D.4.8) can be found in Table IX.

The field of software metrics is not uncontroversial. Many proposed metrics become the subject of debate and dispute concerning whether or not the proposed metric meets the representation condition, that the values assigned by the software measurement faithfully model the relationship that pertains in the real world. By treating the metric as a fitness function, it becomes possible to assess whether the metric really does capture the property of interest; if it does then the optimisation process will produce a sequence of increasingly good solutions. If the solutions do not increase in quality with increases in metric value, then the optimisation will have found a counter example to the representation condition for the metric [217].

Optimisation can also be used to select from among a set of potential metrics, the optimal set that captures properties of interest in a predictive manner [460, 461, 462]. Using this approach a meta metric assesses the accuracy of the prediction, based on a set of metrics that can be easily measured. Typically, this approach is applied to problems where the meta metric can only be measured post production, whereas the metrics used for prediction can be gathered at the outset or early stages of development.

Metrics can also be used to identify programmer style. Lange and Mancoridis [290] applied a GA to identify software developers based on the style metrics.

X. Management

Software engineering management is concerned with the management of complex activities being carried out in different stages of the software life cycle, seeking to optimise both the processes of software production as well as the products produced by this process. Task and resource allocation, scheduling and cost-effort estimation have been among the most frequently studied problems studied in this category. A summary of the papers addressing problems in the area of Management (ACM: D.2.9) are summarized in Table X. The papers can be roughly categorized according to whether they concern project planning activities or whether they create predictive models to provide decision support to software project managers. The following two subsections present the work in each of these two categories.

1) Project planning: Chang *et al.* [100, 101, 102, 103, 105] were the first to use SBSE on Software Management problems. Their early work on search based software project management [100, 101, 105] was among the first papers on SBSE. They introduced the Software Project Management Net (SPMNet) approach for project scheduling and resource allocation, evaluating SPMNet on simulated project data. SPMNet deals with project scheduling and resource allocation. Other early work on SBSE for project management was presented by Aguilar-Ruiz *et al.* [7, 8] , who also advocated the use of a Software Project Simulator (SPS) to evaluate fitness, guiding an evolutionary search for a set of management rules to inform and assist the project manager.

The allocation of teams to work packages in software project planing can be thought of as an application of a bin packing problem [119]. Motivated by this observation, Antoniol *et al.* [33, 35] and Chicano and Alba [17, 22] applied search algorithms to software projects. Antoniol *et al.* applied genetic algorithms, hill climbing and simulated annealing to the problem of staff allocation to work packages. They also considered problems of re-working and abandonment of projects, which are clearly important aspects of most Software Engineering projects. Antoniol *et al.* applied the algorithms to real world data from a large Y2K maintenance project. Chicano and Alba consider the multi objective version of the problem applied to synthetic data. The multiple objectives are combined into a single fitness function using weights for each of the component objectives.

Bouktif *et al.* have used SBSE to consider the management problem of determining the expected quality of software system as a prediction system. Bouktif *et al.* [74] present a GA-based quality model to improve software quality prediction, while Bouktif *et al.* [77] show how the general problem of combining quality experts, modeled as Bayesian classifiers, can be tackled via an SA algorithm customization and Bouktif *et al.* [75] use GA-based method to improve rule set based object-oriented software quality prediction.

The application areas of software project management, scheduling and planning have witnessed a great deal of recent interest from the research community, with recent contributions from a number of authors. Alvarez-Valdes *et al.* [28] used a Scatter Search algorithm for project scheduling problems to minimise project completion duration. This is one of the few applications of scatter search in SBSE. Barreto *et al.* [56] propose an optimisation-based project-staffing algorithm to solve staffing problem. Cortellessa *et al.* [128] describe an optimisation framework to provide decision support for software architects. Hericko *et al.* [238] use a simple gradient-based optimisation method to optimise project team size while minimising project effort. Kapur *et al.* [266] use a GA to provide optimal staffing for product release and best quality to customers under time constraints. Kiper *et al.* [279] apply GA and SA to select optimal subset of V&V activities in order to reduce risk under budget restrictions, thereby linking the problem domains of testing and management.

It is clear that this application area will continue to draw interest and activity from the SBSE community. Though there has been much interest the difficulty of the problem of software project management, there remain a number of unresolved challenges, including:

1) **Robustness**. It may not be sufficient to find a project plan that leads to early completion time. It may be more important to find plans that are robust in the presence of changes. Such a robust plan may be sub-optimal with respect to the completion time objective. This may be a worthwhile sacrifice for greater certainty in the worst case completion time, should circumstances change. These forms of 'robustness trade off' have been widely studied in the optimisation literature [65].
2) **Poor Estimates**. All work on Software project estimation has had to contend with the problem of notoriously

poor estimates [428]. Much of the work on SBSE for project management has implicitly assumed that reliable estimates are available at the start of the project planning phase. This is a somewhat unrealistic assumption. More work is required in order to develop techniques for software project planning that are able to handle situations in which estimates are only partly reliable.

3) **Integration**. Software project management is a top level activity in the software development lifecycle. It draws in the other activities such as design, development, testing, and maintenance. As such, project management is ideally, not an activity that can be optimised in isolation. In order to achieve wider applicability for the SBSE approach to software project management, it will be necessary to develop techniques that can integrate management activities with these other engineering activities, balancing across many competing objectives, relating to aspects of each phase that can impact on the overall objectives of a sound project plan with a realistic but an attractively early completion date.
   Software project management also cannot be conducted in isolation from requirement engineering, since the choice of requirements may affect the feasibility of plans. Therefore, though the requirements gathering and analysis phases typically precede the formulation of management planning this is clearly not desirable once one accepts that the planning phase can be formulated as an optimisation problem.
   Early work on integration by Saliu and Ruhe [412] showed how implementation objectives and requirements objectives could be simultaneously optimised using a multi objective optimisation approach. More work is required to integrate other aspects of the software development process into an optimised software project management activity.

Figure 6 provides a generic schematic overview of SBSE approaches to project planning. Essentially, the approach is guided by a simulation that captures, in abstract form, the conduct of the project for a given plan. A project plan is evaluated for fitness using the simulation. Typically the simulation is a simple queuing simulation that can deterministically compute properties of the project (such as completion time), based upon the plan. The plan involves two aspects: people and tasks. The tasks (usually called work packages) have to be completed by teams. There may be dependencies between the work packages which mean that one cannot start until another is completed. Work packages may also require certain skills, possessed by some staff (and not others), while staff may be assigned to teams.

These details form the basis of the different choices of formulation of the problem studied in the literature. However, all are united by the overall approach, which is to assess fitness of a project plan, using a model of its conduct, with the search space of possible project plans, taking into account different aspects of the real world software project management problem as determined by the problem formulation.

2) Cost Estimation: Software project cost estimation is known to be a very demanding task [428]. For all forms of project, not merely those involving software, project estimation activities are hard problems, because of the inability to 'predict the unpredictable' and the natural tendency to allocate either arbitrary (or zero) cost to unforeseen (and unforeseeable) necessitated activities. The problem of estimation is arguably more acute for software projects than it is for projects in general, because of

1) the inherent uncertainties involved in software development,
2) the comparative youth of the Software Engineering as a discipline and
3) the wide variety of disparate tasks to which software engineering solutions can be applied.

Dolado was the first author to attack software project estimation problems using SBSE. He applied genetic programming to the problem of cost estimation, using a form of 'symbolic regression' [148, 149, 150]. The idea was to breed simple mathematical functions that fit the observed data for project effort (measured in function points). This has the advantage that the result is not merely a prediction system, but also a function that *explains* the behaviour of the prediction system.

Several authors have used GP in cost estimation and quality prediction systems. Evett *et al.* [162] use GP for quality prediction. Liu and Khoshgoftaar [314] also apply GP to quality prediction, presenting two case studies of the approach. This GP approach has been extended, refined and further explored by Khoshgoftaar *et al.* [272, 273, 276, 313, 315]. In all these works, GP evolved predictors are used as the basic for decision support. Other authors have used a combination of GA and GP techniques for estimation as a decision support tool for software managers. Huang *et al.* [248] integrate the grey relational analysis (GRA) with a GA to improve the accuracy of software effort estimation. Jarillo *et al.* [252] apply GA and GP to effort estimation for predicting the number of defects and estimating the reliability of the system. Lokan [316] investigate the performance of GP-based software effort estimation models using a number of fitness functions.

Burgess and Lefley also report results from the application of GP to software project cost estimation [92, 291]. Shan *et al.* [425] compare a grammar-guided GP approach with linear regression in estimation of software development cost. Sheta [430] present two new model structures to estimate the effort required for the development of software projects using GAs and bench-mark them on a NASA software project data set. Shukla [432] present a neuro-genetic approach using a genetically trained neural network (NN) predictor trained to predict resource requirements for a software project based on historical data.

Kirsopp *et al.* [280] also used search techniques in software project cost estimation. Their approach predicts unknown project attributes in terms of known project attributes by seeking a set of near neighbour projects that share similar values for the known attributes. This approach is known as 'Case Based Reasoning' and it is widely used in prediction systems. Case Based Reasoning works well when the existing base of project data is of consistently good quality, but can perform badly where some projects and/or attributes are miss-recorded.

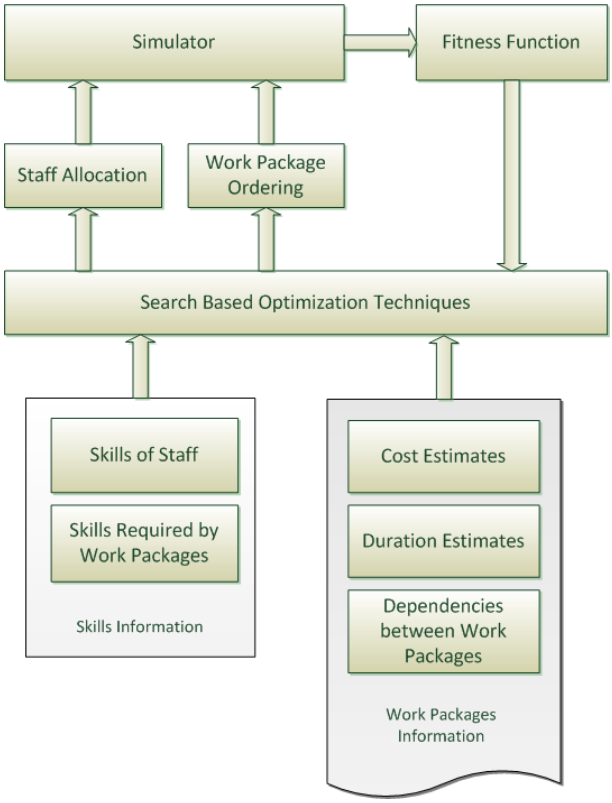


Fig. 6. A Generic Search Based Project Management Scheme

Kirsopp at al. showed that the problem of determining a set of good predictors can be formulated as a feature subset selection problem, to which they applied a hill climbing algorithm. This work was also one of the few in the SBSE literature that has evaluated the properties of the search landscape.

XI. Analysis of Techniques & Applications

Figure 1 showed the trend of growth in publications in SBSE, while Figure 2 showed how the applications areas within Software Engineering have been covered (with a large majority of the work focussing on testing). In this section a further and deeper analysis of the overall area is provided using bar graphs to show the relative frequency of application of optimisation techniques, together with a Formal Concept Analysis to show the relationships between application areas and techniques applied.

Figures 7 and 8 provide results on the choice of search based technique to be applied in SBSE. In Figure 7, all Evolutionary Computation approaches (GA, GP, ES) are grouped together. Also, it was found that some authors simply state that they used an 'Evolutionary Algorithm' or that they use 'Evolutionary Computation' without going into detail. These are all included in Figure 7 under the heading 'EA'. This figure, therefore, covers all the papers in the survey between the census dates 1976 and 2008 (inclusive).

By contrast, Figure 8 reports the results, splitting the EA category into its constituent parts: GA, GP and ES. Different authors mean different things by these terms. By 'ES' we mean an Evolution Strategy approach (with no population and no cross over). By 'GP' we mean any evolutionary approach in which the candidate solution is represented as a program, requiring specific genetic programming operators. By 'GA' we mean any algorithm that the authors described as a Genetic Algorithm, including both binary encodings and other encodings that could not be described as programs (so not GP) and for which there was a population (so not counted as ES). These terms are somewhat fuzzy in the literature on optimisation and this is why we explain how the terms are interpreted in this paper. In particular, there were 54 papers which claimed to use 'Evolutionary Algorithms' but were not specific about which kind of algorithm was used. These papers are categorized in Figure 7 (as 'EA'), but are simply omitted from Figure 8.

Given these terminological difficulties, the figures can only be taken as rough guides. However, one or two observations can be made. The most striking is the prevalence of evolutionary computation (EA, GA, GP, EP) approaches in the SBSE literature. More papers use some form of evolutionary computation than all of the other search based techniques combined. This does not appear to be because of an inherent superiority of the evolutionary approach (though it is very adaptable). Rather, this appears to be more an accident of history and sociology. Much more work is required in order to determine whether evolutionary approaches are, indeed, the most favourable for a particular application. There is very little work on the determination of the most suitable optimisation approach in the literature.

Another striking aspect of the SBSE literature (from the

optimisation point of view) is the comparatively widespread use of hill climbing. This simple local search technique is often derided in the optimisation literature, yet it can be extremely effective and has a number of advantages over more sophisticated algorithms:

1) It is extremely efficient: both quick to implement and fast in execution.
2) Though it may become trapped in a local optima, it can be re-stated multiple times. As such, for problems in which a quick answer is required that is merely 'good enough', HC often serves the purpose; the choice of other techniques may denote something of a 'sledge hammer to crack a nut'.
3) It gives a sense of the landscape structure. Because hill climbing performs a local search and ascends the 'nearest hill to the start point', with multiple restarts, it can be a quick and effective way of obtaining a first approximation to the structure of the landscape.

These properties of Hill Climbing make it well suited to new application areas of SBSE (or indeed for any new optimisation problem). The technique can be used to quickly and reliably obtain initial results, test out a putative fitness function formulation, and to assess the structure of the search landscape. In SBSE, where many new application areas are still being discovered, Hill Climbing denotes a useful tool: providing fast, reliable and understandable initial results. It should be tried before more sophisticated algorithms are deployed.

Figure 9 presents a Formal Concept Analysis (FCA) of the literature on SBSE. FCA [439] is an analysis technique that can be applied to tabular data that report objects, attributes and the binary relationships between them. A 'concept' is a maximal rectangle in the adjacency matrix of objects and attributes. That is, a concept denotes a maximal set of objects that posses a given (also maximal) set of attributes.

The results of FCA are typically displayed as a concept lattice, such as that presented in Figure 9. The lattice exploits certain symmetry properties enjoyed by all concept spaces (the details of which are beyond the scope of this paper). These properties have been shown to hold irrespective of the particular choice of objects and attributes, thereby imbuing FCA with an enduring appeal. In the case of Figure 9, the objectives are application areas and the attributes are the search based optimisation techniques that have been applied to the corresponding application areas. A concept is thus a set of Software Engineering application areas to which a set of search based optimisation techniques have been applied, such that no larger set of areas can be found for the same set of techniques and no larger set of techniques can be found for the same areas.

In the lattice, a concept is denoted by a node. The concepts are related to one another by edges. If a node n_1 is related to a node n_2 (with n_2 higher up the diagram) then this means, in the case of the SBSE lattice of Figure 9, that all the application areas present at concept n_1 are also present at concept n_2 and that all the optimisation techniques present at n_2 are also present at n_1.

It turns out that, for all lattices, there is a unique labeling of nodes, such that an objective and attribute need appear only once in the labeling. In the case of the SBSE lattice, the labels correspond to application areas in Software Engineering and optimisation techniques. An application area appearing at node n, also implicitly appears at all the nodes reachable from n moving up the lattice. By symmetric counter part, an application area that appears at a node m in the lattice also implicitly appears at all the nodes reachable from m, traveling down the lattice.

The lattice for the SBSE literature reveals a few interesting properties of the clustering of application areas and techniques. First, it is clear that the testing application area has had every optimisation technique applied to it in the SBSE literature (because it appears at the bottom of the lattice), while no technique has been applied to every area (indicating that there are still gaps here). Furthermore, four techniques: TS, SQP, MA and EDA have *only* been applied so far in Software Testing. Of those techniques so far explored these are the least widely applied.

It is also clear that the most widely applied techniques are SA and EAs, backing up the findings of Figures 7 and 8. Hill climbing, though popular, has only be applied to Design, Maintenance, Management and Testing. Only EAs have been applied to Agents, while Protocols form an interesting link between PSO and SA (they are the only application areas, apart from the ubiquitous area of testing) to which both PSO and SA have been applied.

The figure can also be read a little like a subsumption diagram. For example, all areas to which IP, HC and ACO have been applied have also had SA applied to them and all these have had EAs applied to them. Reading the relationship in the other direction, all techniques applied to Agents have also been applied to Coding and all these have been applied to Requirements. The reader may also find other relationships in the lattice that are of interest, depending upon the particular areas and techniques that are of interest to them.

XII. How SBSE Reunites Previously Divergent Areas of Software Engineering

In the early development of the field of Software Engineering the nascent field split into different topic areas, with many different disjoint communities focussing on different aspects of the emerging discipline. Of course, this has been natural and necessary evolution of the subject and it was to be expected. However, it has had the draw back of creating silos of activity with few connections between them.

As well as its advantages within areas of Software Engineering, SBSE can also act as a catalyst to remove barriers between subareas, thereby combating the disadvantages of 'silo mentality'. It is interesting to observe how SBSE creates these linkages and relationship between areas in Software Engineering that would otherwise appear to be completely unrelated.

For instance, the problems of Requirements Engineering and Regression Testing would appear to be entirely unrelated topics. Indeed, these two areas of Software Engineering soon developed their own series of conferences, with work on Requirements Engineering tending to appear in the conference

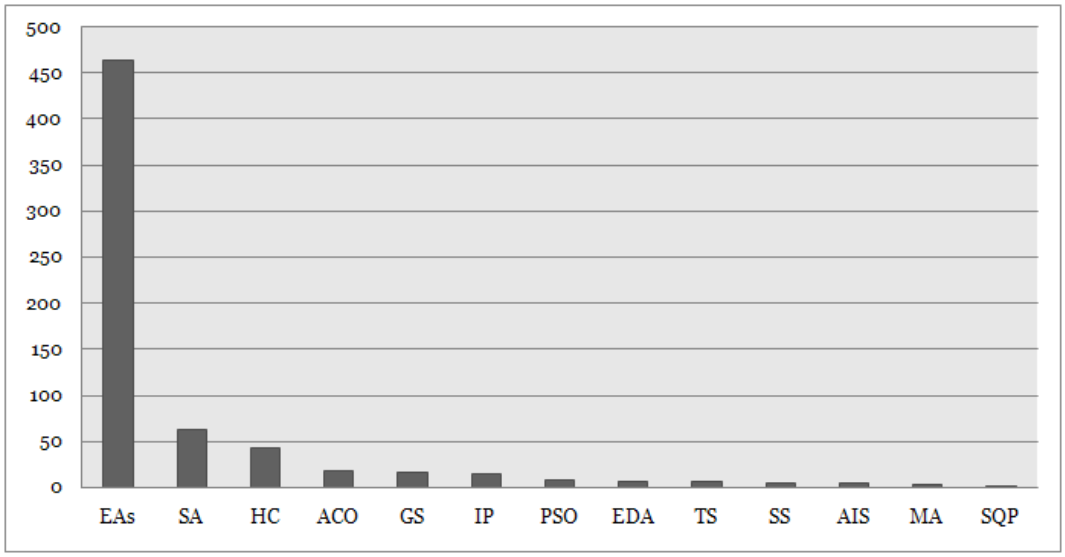


Fig. 7. Numbers of papers using each of the different types of search based optimisation techniques: EAs is a 'catch all' classification for Evolutionary optimisation.

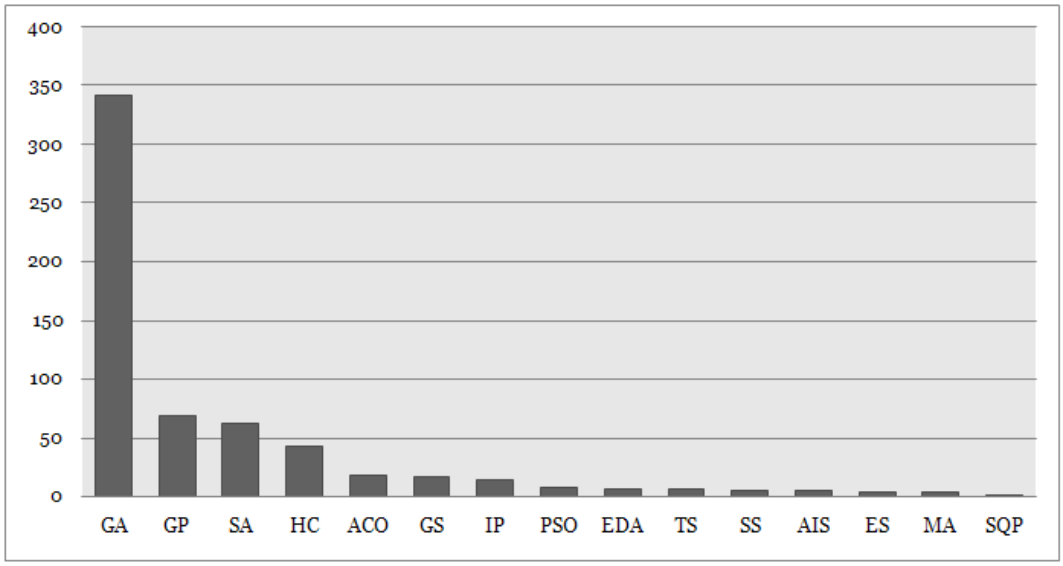


Fig. 8. Numbers of papers using each of the different types of search based optimisation techniques: evolutionary techniques are split into GA, GP and ES.

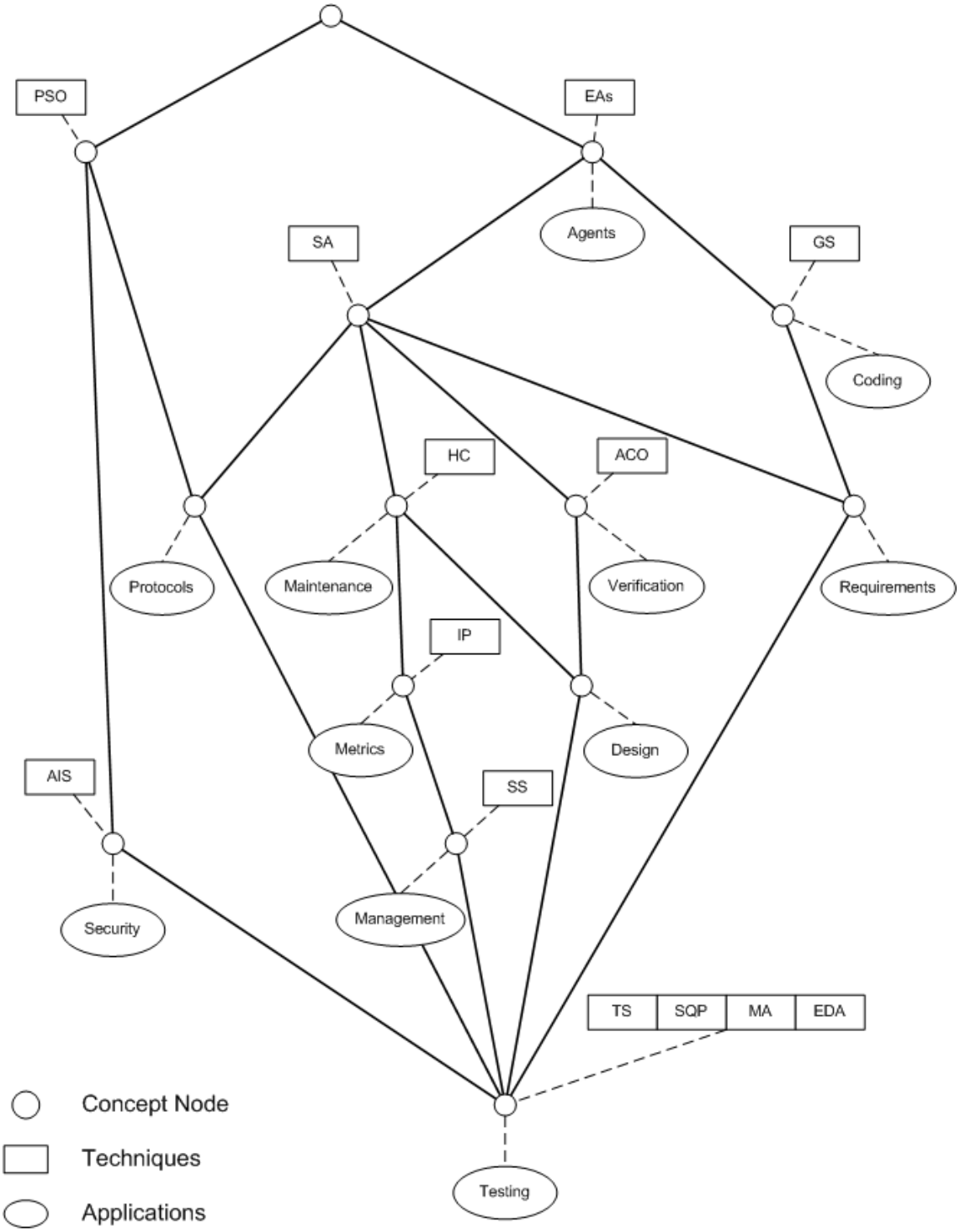
and journal of the same name, while work on regression testing would tend to appear at conferences such as the ACM International Symposium on Software Testing and Analysis and the IEEE International Conference on Software Testing.

However, using SBSE, a clear relationship can be seen between these two problem domains. As optimisation problems they are remarkably similar, though they occur at different phases of the software development process and, typically, researchers working within each topic will form disjoint communities.

Figure 10 illustrates the SBSE-inspired relationship between Requirements Optimisation and Regression Testing. As a selection problem, the task of selecting requirements is closely related to the problem of selecting test cases for regression testing. The difference is that test cases have to cover code in order to achieve high fitness, whereas requirements have to cover customer expectations. In the detail, there will be differences in these two forms of coverage, but as optimisation problems, the similarity is striking: both can be viewed as subset selection problems and also as set cover problems.

When one turns to the problem of prioritization, the similarity is also most striking. Both regression test cases and requirements need to be prioritized. In Requirement Analysis, we seek an order that will ensure that, should development be interrupted, then maximum benefit will have been achieved for the customer at the least cost to the developer; a classic multi objective cost/benefit problem. For test cases, the prioritization must seek to ensure that, should testing be stopped, then maximum achievement of test objectives is achieved with minimum test effort.

This is a very appealing aspect of SBSE. It has the potential to create links and bridges between areas of Software



EAs = Evolutionary Algorithms (including GA, GP and ES)
ACO = Ant Colony Optimization
PSO = Particle Swarm Optimization
EDA = Estimation of Distribution Algorithms
SQP = Sequential Quadric Programming
AIS = Artificial Immune Systems

TS = Tabu Search
SS = Scatter Search Algorithm
MA = Memetic Algorithms
SA = Simulated Annealing
HC = Hill Climbing
GS = Greedy Search
IP = Integer Programming

Fig. 9. Formal Concept Analysis for Techniques & Applications. in the SBSE literature 1976–2008

Engineering that have grown apart over the years, but which submit to similar analysis from the optimisation point of view. Such relationships may lead to exciting new opportunities for cross fertilization between disjoint research communities. These opportunities are a compelling reason for there to be conferences and events that focus on Search Based Software Engineering. The approach clearly has the potential to cut across traditional Software Engineering boundaries.

XIII. Overlooked and Emerging Areas

Some areas of SBSE activity have been considered briefly in the literature and then appear to have been overlooked by subsequent research. This section highlights these areas. That is, topics that have been addressed, shown promising results, but which had attracted neither follow-on studies nor (relatively speaking) many citations. Given the initially patchy nature of work on SBSE and the recent upsurge in interest and activity, these potentially overlooked areas may be worthy of further study.

Furthermore, this survey comes at a time when SBSE research is starting to become widespread, but before it has become mainstream. It is too soon to know whether some of the areas that have been apparently hitherto overlooked, might not simply be emerging areas in which there will be intense activity over the next few years. This section considers both emergent and overlooked ares together; these areas denote either Software Engineering subareas or Optimisation potentialities that remain to be more fully explored.

A. Information Theoretic Fitness

Lutz [318], considered the problem of hierarchical decomposition of software. The fitness function used by Lutz is based upon an information-theoretic formulation inspired by Shannon [426]. The function awards high fitness scores to hierarchies that can be expressed most simply (in information theoretic terms), with the aim of rewarding the more 'understandable' designs. The paper by Lutz is one of the few to use information theoretic measurement as a fitness mechanism. This novel and innovative approach to fitness may have wider SBSE applications.

Recently, Feldt *et al.* [176] also used an information theoretic model, drawing on the observation that the information content of an object can be assessed by the degree to which is can be compressed (this is the so-called Kolmogorov complexity). This recent work may be an indication that information theoretic fitness is not likely to remain an 'overlooked area' for much longer. The authors believe that there is tremendous potential in the use of information theory as a source of valuable fitness for Software Engineering; after all, Software Engineering is an information-rich discipline, so an information theoretic fitness function would seem to be a natural choice.

B. Optimisation of Source Code Analysis

Only a few papers appear to concern source code based SBSE. This is likely to be a growth area, since many source code analysis and manipulation problems are either inherently undecidable or present scalability issues. The source code analysis community has long been concerned with a very rigid model of analysis, in which conservative approximation is the favoured approach to coping with the underlying undecidability of the analysis problem.

However, more recently, Ernst's seminal work on the detection of likely invariants [159], which spawned the widely-used and influential Daikon tool [160] demonstrated that unsound analyses can yield extremely valuable results. The full potential of this observation has yet to be realized. Through the application of SBSE, it will be possible to search for interesting features and to provide probabalistic source code analyses that, like the Daikon work, may not be sound, but would nonetheless turn out to be useful.

A summary of the papers addressing problems related to Coding Tools and Techniques (D.2.3) are summarized in Table V. All of these papers could be regarded as representing an emerging area of optimisation for source code analysis using SBSE. Hart and Shepperd [233] address automatic evolution of controller programs problem by applying GA to improve the quality of the output vector, while Di Penta *et al.* [141, 144] propose a GA-based approach for grammar inference from program examples toward suitable grammars. The grammar captures the subset of the programming language used by the programmer and can be used to understand and reason about programming language idioms and styles.

Jiang *et al.* [257, 258] use search-based algorithms to decompose the program into slices and to search for useful dependence structures. The search problem involves the space of subsets of program slices, seeking those that denote decomposable but disparate elements of code using metaheuristic search and also greedy algorithms. The results showed that, as procedures become larger, there was a statistically significant trend for them to become also increasingly splitable.

C. SBSE for Software Agents and Distributed Artificial Intelligence

Software agents and the general areas known by the term 'Distributed Artificial Intelligence' in the ACM classification system, would seem to provide a rich source of problems for SBSE, particularly those approaches that use population based optimisation. A summary of the papers addressing Distributed Artificial Intelligence (ACM: I.2.11) are summarized in Table XI. As can be seen there is comparatively little work in this area, despite there being some early work by Sinclair and Shami [438], who investigate the relative efficiency of GA and GP to evolve a grid-based food gathering agent. More recently, Haas *et al.* [211] used a GA for parameter tuning of multi-agent systems, while Hodjat *et al.* [244] apply GAs to improve agent-oriented natural language interpreters.

This apparent lack of other previous work[6] is something of a surprise since the nature of multi agent systems seems

[6]The work of Simons and Parmee [436, 437] is categorised in this paper under the topic of 'design', since it is primarily concerned with this application. However, it should be noted that Simons and Parmee [436] also use intelligent agents as part of their search based approach to software design.

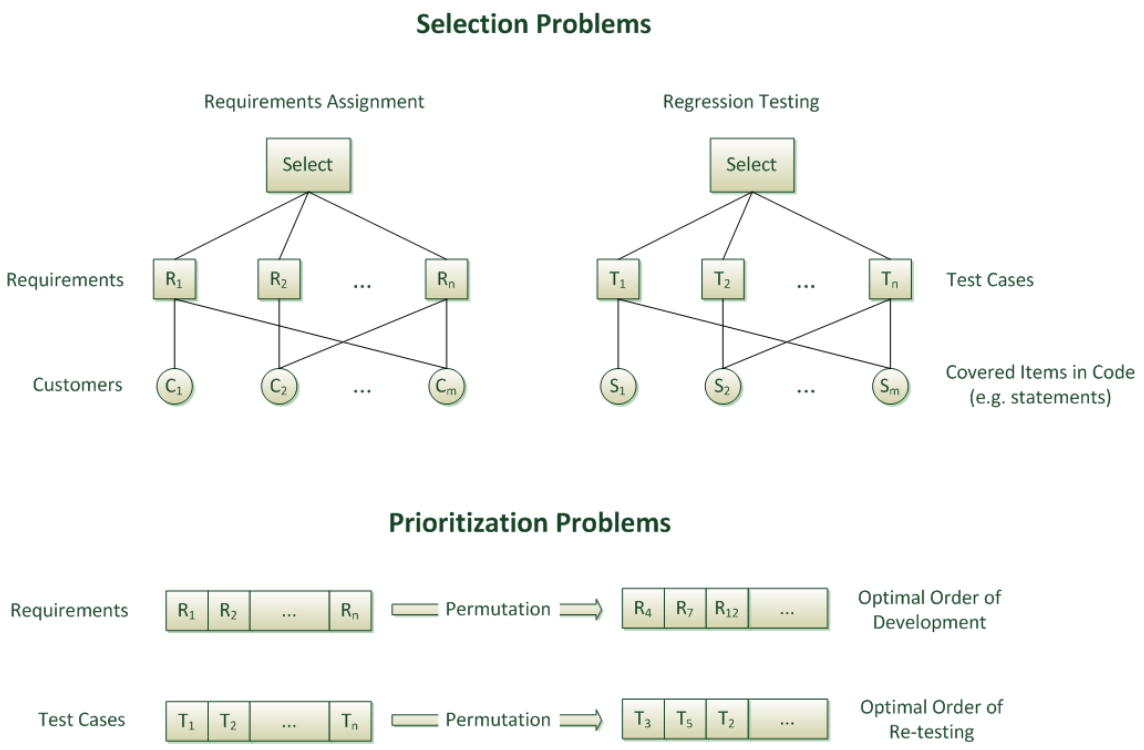


Fig. 10. Requirements Selection & Regression Testing

very closely aligned and amenable to SBSE. That is, an agent based system consists of a population of individuals that interact and share information, seeking to solve common goal. A population based optimisation algorithm also consists of a set of individuals that exchange information trough cross over. Furthermore, co-evolutionary optimisation seems particularly well suited to the Agent Oriented Paradigm; each agent could co-evolve its beliefs, desires and intentions in co-evolutionary co-operation with the others. Alternatively, using competitive co-evolution, it may be possible to identify good agent designs by creating an environment in which they are subjected to evolutionary pressure, using GP to evolve their internal structure.

The authors believe that the potential for SBSE applications in the area of Software Agents is enormous. Recent work[7] [368] demonstrated how an agent can be tested using SBSE techniques. We hope to develop this model of Evolutionary Agents further in future work.

D. Security and Protection

There have been very few papers on the application of SBSE to problems of security. A summary of the papers addressing Security and Protection areas (ACM: K.6.5) are summarized in Table XII. This is sure to change, given the importance of this area of application. The challenge is often to find a way to encode a security problem as a fitness function.

Often security aspects have a decidedly boolean character to them; either there is a security problem is present or it is absent. In order to fully apply SBSE techniques to find security problems, it will be necessary to find a way to formulate fitness functions that offer a guiding gradient towards an optimum.

Some authors have managed to do this. Dozier *et al.* [153] describe how the design of AIS-based Intrusion Detection Systems (IDSs) can be improved through the use of evolutionary hackers in the form of GENERTIA red teams (GRTs) to discover holes found in the immune system. Dozier *et al.* [154] compare a hacker with 12 evolutionary hackers based on Particle Swarm Optimisation (PSO) that have been used as vulnerability analyzers for AIS-based IDSs. Del Grosso *et al.* [133, 134] showed how SBSE can be used to detect buffer overflow vulnerabilities, thereby helping to guard against 'stack smash' attacks.

E. Protocols

Protocol correctness, efficiency, security and cost are all aspects of protocol definitions that can and have been explored using SBSE. Alba and Troya [24] present a first attempt in applying a GA for checking the correctness of communication protocols (expressed as a pair of communicating FSMs). Clark and Jacob [115] use GA in the design and development of Burrows, Abadi and Needham (BAN) protocols optimizing for the tradeoff between protocol security, efficiency and cost. This was subsequently extended by Clark and Jacob [116], who apply GA and SA approaches to the problem addressed in Clark and Jacob [115]. El-Fakih *et al.* [155] use the 0-1 integer linear programming and GAs to solve the message exchange optimisation problem for distributed applications in order to reduce the communication cost. Ferreira *et al.* [180] propose PSO to detect network protocol errors in concurrent

[7]The work is not included in the tables and analysis in this survey since it is published after the census date.

systems. A summary of the papers addressing problems in the area of Network Protocols (ACM: C.2.2) using search based approach are summarized in Table II.

F. Interactive Optimisation

All of the fitness functions so far considered in the literature on SBSE have been fully automated. This seems to be a pre-requisite; fast fitness computation is needed for repeated evaluation during the progress of the search. However, outside the SBSE domain of application, there has been extensive work on fitness functions that incorporate human judgement [184]. This form of search is known as interactive optimisation and it is clearly relevant in many aspects of Software Engineering, such as capturing inherently intuitive value judgements about design preferences [437].

In Software Engineering, interactive optimisation could be used in a number of ways. Many problems may naturally benefit from human evaluation of fitness. For example, in design problems, the constraints that govern the design process may be ill-defined or subjective. It may also be possible to use a search based approach to explore the implicit assumptions in human assessment of solutions. For example, by identifying the building blocks that make up a good solution according to a human fitness evaluation, it may be possible to capture otherwise implicit design constraints and desirable features.

The key problem with any interactive approach to optimisation lies in the requirement to repeatedly revert to the human for an assessment of fitness, thereby giving rise to possible fatigue and learning-effect bias. If this fatigue problem can be overcome in the Software Engineering domain (as it has in other application domains) then interactive optimisation offers great potential benefits to SBSE.

Harman [214] provides an overview of Search Based Software Engineering for problems in program comprehension, which includes ways in which interactive evolution might be applied in problems relating to code understanding.

G. Hybrid Optimisation Algorithms

Many problems have unpredictable landscapes. These can respond well to hybrid approaches. Though there has been work on the combination of existing non-search based algorithms [227], there has been little work on combinations of search algorithms [134, 323].

A better understanding of search landscapes may suggest the application of hybrid search techniques, which combine the best aspects of existing search algorithms. Hybrids are natural in search. For example, it is well known that, for any search application, it makes sense to conclude a run of a genetic algorithm with a simple hill climb from each solution found. After all, hill climbing is cheap and effective at locating the nearest local optima; why not apply it to the final population?

There are also other possible combinations of search algorithms that have been studied in applications to other engineering areas, for example simulated annealing and genetic algorithms have been combined to produce an annealing GA [448]. Estimation of Distribution Algorithms (EDAs) [367] are another form of hybrid search, in which the algorithm's behaviour is adapted as the search proceeds.

H. On Line Optimisation

All applications of SBSE of which the authors are aware concern what might be termed 'static' or 'offline' optimisation problems. That is, problems where the algorithm is executed off line in order to find a solution to the problem in hand. This is to be contrasted with 'dynamic' or 'on line' SBSE, in which the solutions are repeatedly generated in real time and applied during the lifetime of the execution of the system to which the solution applies.

The static nature of the search problems studied in the existing literature on SBSE has tended to delimit the choice of algorithms and the methodology within which the use of search is applied. Particle Swarm Optimisation [507] and Ant Colony Optimisation [151] techniques have not been widely used in the SBSE literature. These techniques work well in situations where the problem is rapidly changing and the current best solution must be continually adapted.

For example, the paradigm of application for Ant Colony Optimisation is dynamic network routing, in which paths are to be found in a network, the topology of which is subject to continual change. The ants lay and respond to a pheromone trail that allows them quickly to adapt to network connection changes.

It seems likely that the ever changing and dynamic nature of many Software Engineering problems would suggest possible application areas for Ant Colony Optimisation and Particle Swarm Optimisation techniques. It is somewhat surprising that highly adaptive search techniques like Ant Colony Optimisation have yet to be applied widely in SBSE. Perhaps distributed, service oriented and agent oriented Software Engineering paradigms will provide additional candidate application areas for ant colony and particle swarm optimisation.

I. SBSE for Non Functional Properties

There has been much work on stress testing [14, 80, 81, 186, 187, 189, 332] and temporal tetsing [13, 14, 15, 16, 147, 201, 202, 203, 204, 205, 381, 449, 480, 481, 481, 482, 483], but far less on other non functional properties such as heat dissipation and power consumption [264, 488] and thermal properties such as temperature and heat dissipation [264]. The problem of (Quality of Service (QoS) introduced by Canfora *et al.* [95], also denotes an area of non–functional optimisation in Software Engineering which has recently witnessed an upsurge in activity and interest [250, 319, 446, 504, 505].

It seems likely that the drive to ever smaller devices and to massively networked devices will make these issues far more pressing in future, thereby engendering more research in this area. Afzal *et al.* [6] provide a detailed in–depth survey of approaches to testing non–functional requirements, to which the readers is refereed for a more detailed treatment of this area.

J. Multi Objective Optimisation

Software engineering problems are typically multi objective problems. The objectives that have to be met are often competing and somewhat contradictory. For example, in project

planning, seeking earliest completion time at the cheapest overall cost will lead to a conflict of objectives. However, there does not necessarily exist a simple tradeoff between the two, making it desirable to find 'sweet spots' that optimise both.

Suppose a problem is to be solved that has n fitness function, $f_1, \ldots, f_n$ that take some vector of parameters $\overline{x}$. One simple-minded way to optimise these multiple objectives is to combine them into a single aggregated fitness, F, according to a set of coefficients, $c_i, \ldots, c_n$:

$$F = \sum_{i=1}^{n} c_i f_i(\overline{x})$$

This approach works when the values of the coefficients determine precisely how much each element of fitness matters. For example, if two fitness functions, f_1 and f_2 are combined using

$$F = 2 \cdot f_1(\overline{x}) + f_2(\overline{x})$$

then the coefficients $c_1 = 2, c_2 = 1$ explicitly capture the belief that the property denoted by fitness function f_1 is twice as important as that denoted by fitness function f_2. The consequence is that the search may be justified in rejecting a solution that produces a marked improvement in f_2, if it also produces a smaller reduction in the value of f_1.

Most work on SBSE uses software metrics in one form or another as fitness functions [217]. However, the metrics used are often those that are measured on an *ordinal scale* [429]. As such, it is not sensible to combine these metrics into an aggregate fitness in the manner described above.

The use of Pareto optimality is an alternative to aggregated fitness. It is superior in many ways. Using Pareto optimality, it is not possible to measure 'how much' better one solution is than another, merely to determine whether one solution is better than another. In this way, Pareto optimality combines a set of measurements into a single ordinal scale metrics, as follows:

$$F(\overline{x_1}) \geq F(\overline{x_2}) \Leftrightarrow \forall i. f_i(\overline{x_1}) \geq f_i(\overline{x_2})$$

and, for strict inequality:

$$F(\overline{x_1}) > F(\overline{x_2})$$
$$\Leftrightarrow$$
$$\forall i. f_i(\overline{x_1}) \geq f_i(\overline{x_2}) \quad \wedge \quad \exists i. f_i(\overline{x_1}) > f_i(\overline{x_2})$$

Thus, under Pareto optimality, one solution is better than another if it is better according to at least one of the individual fitness functions and no worse according to all of the others. Under the Pareto interpretation of combined fitness, no overall fitness improvement occurs no matter how much almost all of the fitness functions improve, should they do so at the slightest expense of any one of their number.

When searching for solutions to a problem using Pareto optimality, the search yields a set of solutions that are non-dominated. That is, each member of the non-dominated set is no worse than any of the others in the set, but also cannot be said to be better. Any set of non-dominated solutions forms a Pareto front.

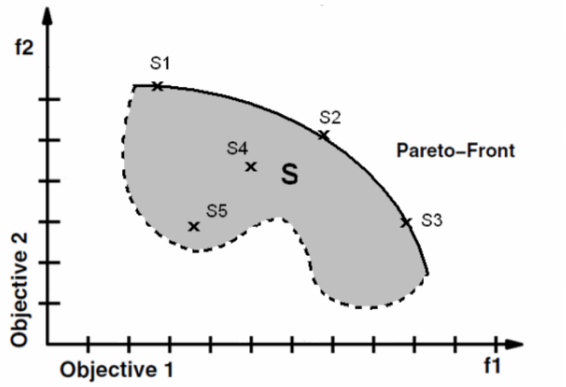


Fig. 11. Pareto Optimality and Pareto Fronts

Consider Figure 11, which depicts the computation of Pareto optimality for two imaginary fitness functions (objective 1 and objective 2). The longer the search algorithm is run the better the approximation becomes to the real Pareto front. In the figure, points $S1$, $S2$ and $S3$ lie on the pareto front, while $S4$ and $S5$ are dominated.

Pareto optimality has many advantages. Should a single solution be required, then coefficients can be re-introduced in order to distinguish among the non-dominated set at the current Pareto front. However, by refusing to conflate the individual fitness functions into a single aggregate, the search is less constrained. It can consider solutions that may be overlooked by search guided by aggregate fitness.

The approximation of the Pareto front is also a useful analysis tool in itself. For example, it may contain knee points, where a small change in one fitness is accompanied by a large change in another. These knee points denote interesting parts of the solution space that warrant closer investigation.

Recently, research on SBSE has started to move from single objective formulations to multi objective formulations, with an increasing focus on Pareto optimal optimisation techniques. Recent work has produce multi objective formulations of problems in many application areas within Software Engineering including requirements [181, 508], testing [133, 161, 230], quality assurance [275], refactoring [222] and project management [22].

K. Co-Evolution

In Co-Evolutionary Computation, two or more populations of solutions evolve simultaneously with the fitness of each depending upon the current population of the other. The idea, as so far applied in SBSE work, is to capture a predator-prey model of evolution, in which both evolving populations are stimulated to evolve to better (i.e. more fit) solutions.

Adamopoulos *et al.* [1] were the first to suggest the application of co-evolution in Software Engineering, arguing that this could be used to evolve sets of mutants and sets of test cases, where the test cases act as predators and the mutants as their prey. Arcuri *et al.* use co-evolution to evolve programs and their test data from specifications [36, 38] using co-evolution.

Arcuri [36], Arcuri and Yao [41] also developed a co-evolutionary model of bug fixing, in which one population essentially seeks out patches that are able to pass test cases, while test cases can be produced from an oracle in an attempt to find the shortcomings of a current population of proposed patches. In this way the patch is the prey, while the test cases, once again, act as predators. The approach assumes the existence of a specification to act the oracle.

Co-evolution can also be conducted in a co-operative manner, though this remains, hitherto, explored in SBSE work. It is likely to be productive in finding ways in which aspects of a system can be co-evolved to work better together and, like the previously studied competitive co-evolutionary paradigm, offers great potential for further application in SBSE.

Many aspects of Software Engineering problems lend themselves to a co-evolutionary model of optimisation because software systems are complex and rich in potential populations that could be productively co-evolved (using both competitive and co-operative co-evolution). As with traditional SBSE, it is testing where the analogy is perhaps clearest and most easily applied, which may be why this area has already been considered in the literature. However, there are many other Software Engineering populations that could be evolved, possibly in co-evolution with others.

For example: components, agents, stakeholder behaviour models, designs, cognitive models, requirements, test cases, use cases and management plans are all important aspects of software systems for which optimisation is an important concern. Though all of these may not occur in the same systems, they are all the subject of change, and should a suitable fitness function be found, can therefore be evolved. Where two such populations are be evolved in isolation, but participate in the same overall software system, it would seem a logical 'next step', to seek to evolve these populations together; the fitness of one is likely to have an impact on the fitness of another, so evolution in isolation may not be capable of locating the best solutions. Like the move from single to multiple objectives, the migration from evolution to co-evolution offers the chance to bring theory and real world reality one step closer together.

XIV. Future Benefits to be Expected from Optimisation in Software Engineering

This section briefly reviews some of the benefits that can be expected to come from further development of the field of search based Software Engineering. These benefits are pervading, though often implicit, themes in SBSE research. To borrow the nomenclature of Aspect Oriented Software Development, these are the 'cross cutting concerns' of the SBSE world; advantages that can be derived from almost all applications at various points in their use.

A. Generality and Applicability

One of the striking features of the SBSE research programme that emerges from this survey is the wide variety of different Software Engineering problems to which SBSE has been applied. Clearly, testing remains the predominant 'killer application', with 59% of all SBSE papers targeting various aspects of testing. However, as the survey reveals, there are few areas of Software Engineering activity to which search based optimisation remains unapplied.

This generality and applicability comes for the very nature of Software Engineering. The two primary tasks that have to be undertaken before a search based approach can be applied to a Software Engineering problem are the definition of a *representation* of the problem and the *fitness function* that captures the objective or objectives to be optimised. Once these two tasks are accomplished it is possible to begin to get results from the application of many search based optimisation techniques.

In other engineering disciplines, it may not be easy to represent a problem; the physical properties of the engineering artifact may mean that simulation is the only economical option. This puts the optimisation algorithm at one stage removed from the engineering problem at hand. Furthermore, for other engineering disciplines, it may not be obvious how to measure the properties of the engineering artifact to be optimised. Even where the measurements required may be obvious, it may not be easy to collect the readings; once again the physical properties of the engineering materials may be a barrier to the application of optimisation techniques.

However, software has *no* physical manifestation. Therefore, there are fewer problems with the representation of a software artifact, since almost all software artifacts are, by their very nature, based on intangible 'materials' such as information, processes and logic. This intangibility has made many problems for Software Engineering. However, by contrast, within the realm of SBSE, it is a significant advantage. There are few Software Engineering problems for which there will be no representation, and the readily available representations are often ready to use 'out of the box' for SBSE.

Furthermore, measurement is highly prevalent in Software Engineering, with a whole field of research in software metrics that has spawned many conferences and journals. Therefore, it is also unlikely that the would-be search based software engineer will find him or herself bereft of any putative fitness function. Indeed, it has been argued that all metrics are also fitness functions, waiting to be applied in SBSE [217].

For these reasons, it is likely that there will be a rapid growth in the breadth of SBSE research. The growth trend revealed by Figure 1 in this survey is very likely to continue and authors are likely to continue to find ways to bring new Software Engineering subareas within the remit of SBSE.

B. Scalability

One of the biggest problems facing software engineers is that of scalability of results. Many approaches that are attractive and elegant in the laboratory, turn out to be inapplicable in the field, because they lack scalability.

One of the attractions of the search based model of optimisation is that it is *naturally* parallelizable. Hill climbing can be performed in parallel, with each climb starting at a different point [323]. Genetic algorithms, being population based, are also naturally parallel; the fitness of each individual

can be computed in parallel, with minimal overheads. Search algorithms in general and SBSE in particular, therefore offer a 'killer application' for the emergent paradigm of ubiquitous user-level parallel computing.

Notwithstanding a breakthrough in quantum computation technology, it seems likely that future improvements in processing speed are likely to be based on increasing parallelism. Already, limited parallelism is widely used in desktop computing and the importance of the drive toward superfast parallel computers is recognised at the highest levels. This trend towards greater parallelism, the need for scalable Software Engineering and the natural parallelism of many SBSE techniques all point to a likely significant development of parallel SBSE to address the issue of Software Engineering scalability.

C. Robustness

In some Software Engineering applications, solution robustness may be as important as solution functionality. For example, it may be better to locate an area of the search space that is rich in fit solutions, rather than identifying an even fitter solution that is surrounded by a set of far less fit solutions.

In this way, the search seeks stable and fruitful areas of the landscape, such that near neighbours of the proposed solution are also highly fit according to the fitness function. This would have advantages where the solution needs to be not merely 'good enough' but also 'strong enough' to withstand small changes in problem character [65].

Hitherto, research on SBSE has tended to focus on the production of the fittest possible results. However, many application areas require solutions in a search space that may be subject to change. This makes robustness a natural property to which the research community could and should turn its attention.

D. Feedback and Insight

False intuition is often the cause of major error in software engineering, leading to misunderstood specifications, poor communication of requirements and implicit assumptions in designs. SBSE can address this 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 search based approach. It has been widely observed that search techniques are good at producing unexpected answers. For example, evolutionary algorithms have led to patented designs for digital filters [414] and the discovery of patented antenna designs [308]. Automated search techniques will effectively work in tandem with the human, in an iterative process of refinement, leading to better fitness functions and thereby, better encapsulation of human assumptions and intuition.

XV. Summary

This paper has provided a detailed survey and review of the area of Software Engineering activity that has come to be known as Search Based Software Engineering (SBSE). As the survey shows, the past five years have witnessed a particularly dramatic increase in SBSE activity, with many new applications being addressed.

The paper has identified trends in SBSE research, providing data to highlight the growth in papers and the predominance of Software Testing research. It also indicates that other areas of activities are starting to receive significant attention: Requirements, Project Management, Design, Maintenance and Reverse Engineering, chief among these. The paper also provides a detailed categorization of papers, tabulating the techniques used, the problems studied and the results presented in the literature to date. This detailed analysis has allowed us to identify some missing areas of activity, some potential techniques that have yet to be applied and emerging areas.

The future of SBSE is bright; there are many areas to which the techniques associated with SBSE surely apply, yet have yet to be considered. In existing areas of application the results are strongly encouraging. New developments emanating from the optimisation community will present exciting new possibilities, while developments in Software Engineering will present interesting new challenges. If Software *Engineering* really is an engineering discipline (as the authors strongly believe it to be), then surely SBSE is a very natural consequence of the very existence of such an engineering discipline. After all, is not *optimisation* what engineering is all about?

XVI. Acknowledgement

This analysis and survey paper covers over 500 papers and seeks to capture all work on SBSE up to an including the 31st December 2008. It has been an undertaking requiring considerable time and effort. The authors are deeply indebted to numerous colleagues from the growing international SBSE community who have reviewed and commented upon earlier drafts of this paper and provided additional details regarding their work and that of others, as well as highlighting corrections and omissions. The authors are also very grateful for the support of the collaborators of the EPSRC–funded project SEBASE and the EU–funded project EvoTest. Funding from these two large projects (EP/D050863 and IST-33472 respectively) provided part financial support for the work undertaking in the production of this paper.

TABLE II. Papers addressing activities related to Network Protocols

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Alba and Troya [24]	1996	Checking the correctness of communication protocols	Detect deadlock and useless states or transitions	String	GA	Synthetic & Real
El-Fakih *et al.* [155]	1999	Deriving protocols for distributed applications	Minimise communication cost (minimise number of messages to be exchanged)	Vector	0-1 integer linear programming, GA	Synthetic
Clark and Jacob [115]	2000	Protocol synthesis	Optimise Correctness, cost and efficiency	String	GA	Synthetic
Clark and Jacob [116]	2001	Synthesis of secure protocols	Optimise trade-off between security, efficiency and cost	Integer array (SA), bit string (GA)	SA, GA	Synthetic
Ferreira *et al.* [180]	2008	Detecting protocol errors	Detect deadlock violations	Graph	PSO	Synthetic

TABLE III. Papers addressing activities related to Requirements/Specifications

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Bagnall *et al.* [47]	2001	Requirements Selection and optimisation	Maximise customers' satisfaction	Bit string	GS, HC, SA	Synthetic
Feather and Menzies [166]	2002	Requirements Selection and optimisation	Maximise benefit, Minimise cost	?	SA	Real
Feather *et al.* [167]	2004	System design optimisation	Maximise benefit, Minimise cost	Bit string	SA	Real
Greer and Ruhe [200]	2004	Requirements Selection and optimisation	Minimise total penalty and Maximise total benefit (weighted sum of the two)	Bit string	GA	Real
Baker *et al.* [48]	2006	Requirements Selection and optimisation	Maximise value	Bit string	SA, GS	Real
Feather *et al.* [168]	2006	Visualization Techniques to present Requirements status	-	-	SA	Real
Harman *et al.* [229]	2006	Feature subset selection	Maximise total value	Bit string	GS	Real
Zhang *et al.* [508]	2007b	Requirement satisfaction for the Next Release Problem (NRP)	Maximise value, Minimise cost	Bit string	GA	Synthetic
Finkelstein *et al.* [181]	2008	Fairness analysis in requirements assignments	Maximise each stakeholder's possible satisfaction	Bit string	MOOA (NSGA-II)	Synthetic & Real
Jalali *et al.* [251]	2008	Requirements Decisions optimisation	Maximise the number of attainable requirements, Minimise cost	Bit string	GS	Real
Cortellessa *et al.* [129]	2008b	Automated selection of COTS components	Minimise the cost while assuring the satisfaction of the requirements	Vector	IP (LINGO based)	Synthetic
Zhang *et al.* [509]	2008	overview of existing work and challenges on search based requirements optimisation	-	Bit string	MOOA (NSGA-II)	Synthetic & Real

TABLE IV. Papers addressing activities related to Design Tools and Techniques

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Feldt [169]	1998	Software fault tolerance	Develop multiple software variants	Tree	GP	Real
Feldt [171]	1998	Software fault tolerance	Develop multiple diverse software versions	Tree	GP	Real
Feldt [172] N	1998	Software fault tolerance	Maximise the software program diversity	Tree	GP	Real
Monnier *et al.* [366]	1998	Development of scheduling module for a real-time system	Finding a feasible solution within the time constraints	String	GA	benchmark problems
Feldt [174]	1999	Knowledge acquirement in early software development phases	Prioritize requirements and explore design trade-off	Tree	GP	Real
Lutz [318]	2001	Hierarchical architecture decomposition	Minimise complexity	Tree	GA	Synthetic
Feldt [175]	2002	Interactive software development workbench	New feature and knowledge acquirement	Tree	EA (Biomimetic Algorithms)	Real
Antoniol and Di Penta [31]	2003	Library Miniaturization	Minimise inter-library dependencies; Minimise the number of objects linked by applications	bit-matrix	GAs	Real (Open Source)
O'Keeffe and Ó Cinnéide [371] N	2003	Automated OO design improvement	Minimise rejected, duplicated and unused methods and featureless classes, Maxmise abstract classes	?	SA	?
Stephenson *et al.* [443]	2003	Automatic programming, compiler optimisation	Minimise code execution time (the fastest code is the fittest)	Tree	GP	Real
Canfora *et al.* [94]	2004	QoS-aware service composition	Maximise QoS attributes (availability, reliability); Minimise cost and response time	Integer array	GA	Synthetic
Khoshgoftaar *et al.* [275]	2004	Design and Implementation (improvement of software reliability / quality assurance)	Maximise MOM performance at four cutoff percentiles; Minimise tree size (bloat control fitness function)	Tree	GP, MOOA	Real
Khoshgoftaar *et al.* [274]	2004	Design and Implementation (improvement of software reliability / quality assurance)	Maximise MOM performance at four cutoff percentiles; Minimise tree size (bloat control fitness function)	Tree	GP, MOOA	Real
O'Keeffe and Ó Cinnéide [372]	2004	Automated OO Design Improvement	Minimise rejected, duplicated and unused methods and featureless classes, Maxmise abstract classes	?	SA	Synthetic
Canfora *et al.* [95]	2005	QoS of Composite Services	Maximise QoS attributes (availability, reliability); Minimise cost and response time	Integer array	GA, IP	Synthetic
Canfora *et al.* [96]	2005	QoS-aware replanning of Composite Services	Maximise QoS attributes (availability, reliability); Minimise cost and response time	Integer array	GA	Real
Cao *et al.* [98]	2005b	Web service selection	Minimise the overall cost of each execution path	Integer vector	GA	Synthetic
Cao *et al.* [97]	2005a	Web service selection	Minimise the overall cost	Integer vector	GA	Synthetic
Lucas and Reynolds [317]	2005	Learning Deterministic Finite Automata	Maximise correctness of classification	Binary String	HC	Synthetic & Real

Continued on next page

TABLE IV. Papers on Design Tools and Techniques – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Amoui *et al.* [29]	2006	OO Software architecture design	Optimise metrics and find the best sequence of transformations	?	GA	?
Aversano *et al.* [45]	2006	Design of service composition	Maximise Recall for outputs; Maximise Precision	Tree	GP	Real
Sheu and Chuang [431]	2006	Development of scheduling module for a real-time system	Find a feasible solution within the time constraints	String	GA	Synthetic (via simulation)
Simons and Parmee [434]	2006	Design Comprehension	Maximise Cohesion, Minimise Coupling	Object-based	GA, EP, NSGA-II	Real
Yang *et al.* [498]	2006	Software integration	Minimise software risk	Binary string	GA	Real
Zhang *et al.* [504]	2006	Web services selection	Improve QoS	Matrix	GA	Synthetic
Jaeger and Mühl [250]	2007	Web services selection	Improve the QoS	Vector	GA	Synthetic
Simons and Parmee [435]	2007	Object-oriented conceptual software design	Maximise the Cohesiveness of Methods (COM) metric and number of class	Binary string	GA MOOA (NSGA-II)	Real
Su *et al.* [446]	2007	Web services selection	Improve QoS	Matrix	GA	Synthetic
Zhang *et al.* [505]	2007a	Web services selection	Improve QoS	Matrix	GA	Synthetic
Arcuri *et al.* [43]	2008	Non-functional property optimisation	Minimise non-functional property and error	Tree	GP, MOOA (SPEA2)	Synthetic
Barlas and El-Fakih [55]	2008	Distributed system design	Optimise the delivery to multiple clients by multiple servers	String	GA	Synthetic (Simulation)
Bhatia *et al.* [66]	2008	Reusable Software Component Retrieval	Generate rules for classifying components	Graph	ACO	-
Bowman *et al.* [78]	2008	Class responsibility assignment (OO design)	Maximise Cohesion, Minimise Coupling	Integer string	MOGA, RS, GA, HC	Synthetic
Chardigny *et al.* [107]	2008b	Architecture extraction for OO systems	Improve the quality and the semantic correctness of the architecture	?	?	-
Chardigny *et al.* [106]	2008a	Architecture extraction for OO systems	Improve the quality and the semantic correctness of the architecture	?	SA	Real
Desnos *et al.* [139]	2008	Automatic component substitution	Optimise software reuse and evolution	Tree	BA, BBA	Synthetic
Goldsby and Cheng [197]	2008b	Software behavioral model generation	Identify multiple behavioral models and satisfy functional properties	-	digital evolution (Avida-based)	Real
Goldsby and Cheng [196]	2008a	Software behavioral model generation	Identify multiple behavioral models and satisfy functional properties	-	digital evolution (Avida-based)	Synthetic
Goldsby *et al.* [198]	2008	Software behavioral model generation and satisfy functional properties	Identify multiple behavioral models and satisfy functional properties	-	digital evolution (Avida-based)	Real
Ma and Zhang [319]	2008	Web service selection	Improve the QoS	Matrix	GA	Synthetic
Räihä [383]	2008a	Automated architecture design	Improve efficiency, modifiability and complexity	a collection of supergenes	GA	Synthetic

Continued on next page

TABLE IV. Papers on Design Tools and Techniques – *continued from previous page*

Authors [Ref]	**Year**	**Activity**	**Objective / Fitness**	**Representation method**	**Search technique**	**Test problems**
Räihä [384]	2008b	Automated architecture design	Improve efficiency, modifiability and complexity	a collection of supergenes	GA	Synthetic
Räihä *et al.* [386]	2008a	Automated architecture design	Improve efficiency, modifiability and complexity	a collection of supergenes	GA	Synthetic
Räihä *et al.* [387]	2008b	Automating CIM-to-PIM model transformations	Improve efficiency, modifiability and complexity	a collection of supergenes	?	?
Sharma and Jalote [427]	2008	Deploying software components	Maximise performance	?	Heuristics	Synthetic
Simons and Parmee [436]	2008	Software design supporting	Minimise design coupling, Maximise cohesion of classes	String	MOOA (NSGA-II)	Synthetic
Simons and Parmee [437]	2008	Conceptual Software Design	Maximise cohesion of classes; Minimise coupling between classes	Object-based	MOOA (NSGA-II)	Real
Vijayalakshmi *et al.* [459] N	2008	Component selection in software development	?	?	GA	?

TABLE V. Papers addressing activities related to Coding Tools and Techniques

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Hart and Shepperd [233]	2002	Automatic evolution of controller programs	Maximise the quality of the output vector generated	String	GA	Real
Jiang *et al.* [257]	2007b	Dependence analysis	Maximise code coverage and Minimise the degree of overlap between the slices	String	HC, GA, GS	Real
Di Penta *et al.* [144]	2008	Grammar inference	automatic evolution from grammar fragments to target grammar	String	GA	Real
Hoste and Eeckhout [246]	2008	Compiler optimisation	find a pareto optimal trade-off among metrics (total execution time, compilation time, code size, energy comsumption)	?	MOGA (improved SPEA)	Benchmarks
Jiang *et al.* [258]	2008	Automatic support for procedure splitability analysis	Minimise the overlap of slices representing procedure components	Binary Matrix	GS	Real (open source)

TABLE VI. Papers addressing Software/Program Verification

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Minohara and Tohma [357]	1995	Model checking software reliability growth models	Minimise the errors	Bit string	GA	Real
Godefroid [194]	1997	Model checking for state space of concurrent systems	Detect deadlocks and assertion violations	?	State-less search algorithm (VeriSoft-based)	Real
Alba and Chicano [20]	2007	Model checking for safety properties	find optimal errors trails in faulty concurrent system	Graph	ACO	Real
Alba and Chicano [21]	2007	Model checking for safety errors	Detect errors within low amount of memory and CPU time	Graph	ACO	Real
Alba and Chicano [19]	2007	Model checking (Refutation of safety properties)	Find deadlock states	Graph	ACO	Real
Johnson [259]	2007	Model checking	Maximise the number of program statements that are satisfied	List	ES	Synthetic
Kiper *et al.* [279]	2007	V&V Process for Critical Systems	Maximise the chances of mission success	?	SA, GA	-
Afzal and Torkar [3]	2008	Software Reliability Growth Modeling	Measure the suitability of GP evolved SRGM	Tree	GP	-
Afzal and Torkar [2]	2008	Software Reliability Growth Modeling	measure the adaptability and predictive accuracy of GP evolved model	Tree	GP	Real
Afzal *et al.* [5]	2008	Software Reliability Growth Modeling	Measure the adaptability and predictive accuracy of GP evolved model	Tree	GP	Real
Alba *et al.* [25]	2008	Model checking for finding deadlock in concurrent program	Detecting the shortest paths that lead to deadlocks	Graph	GA	Real
Chicano and Alba [112]	2008	Model Checking for Liveness property	Discover liveness errors; Minimise the required resources	Graph	ACO	Real
Chicano and Alba [111]	2008	Model checking for safety property	Discover safety property violations	Graph	ACO	Real
Chicano and Alba [113]	2008	Model checking for Liveness property	Improve the efficacy and efficiency of searching for liveness property violations	Graph	ACO	Real
He *et al.* [237]	2008	Verification and generation of programs	?	String of verified components	GP	Synthetic
Hsu *et al.* [247]	2008	Software Reliability Growth Modeling	Maximise LSE and MLE	Floating-point	GA	Real
Katz and Peled [269]	2008	Model checking	Automatic generation of concurrent programs to detect the property violations	Tree	GP	Synthetic
Katz and Peled [268]	2008	Model checking	Automatic generation of concurrent programs to detect the property violations	Tree	GP	Synthetic
Shyang *et al.* [433]	2008	Model checking	Detect the locating of deadlocks	Bit string	GA	Real
Wang *et al.* [467]	2008	Testing resource allocation	Maximise software reliability, Minimise the cost	?	MOOA (NSGA-II, MODE)	Synthetic

TABLE VII. Papers addressing Testing and Debugging activities

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Miller and Spooner [355]	1976	Test data generation	Maximise coverage?	Input Vector	Integer Programming	From literature
Fischer [182] N	1977	Test case selection	?	?	Integer Programming	?
Fischer *et al.* [183] N	1981	regression testing	?	?	Integer Programming	?
Hartmann and Robson [234]	1989	Automatic selective revalidation	Overview and extensions of current strategies	-	IP	-
Hartmann and Robson [235]	1990	Automatic selective revalidation	Overview of current strategies	-	Integer Programming	-
Hartmann and Robson [236]	1990	Automatic selective revalidation	Overview of current strategies	-	Integer Programming	-
Korel [281]	1990	Test data generation				
Xanthakis *et al.* [493] N	1992	Testing	?	?	GA	?
Schoenauer and Xanthakis [415]	1993	Test data generation	Maximise coverage of selected substructure	Input Vector	GA	Synthetic
Schultz *et al.* [416]	1993	Test scenario generation	Identify maximum faults	String	GA	Real
Davies *et al.* [131]	1994	Test case generation	Generating optimum path	?	GA	Real
Pei *et al.* [380]	1994	Test data generation	Maximise the selected path coverage	Input Vector	GA	Synthetic
Jones *et al.* [260]	1995	Test data generation	Maximise branch coverage	Input Vector	GA	Synthetic
Jones *et al.* [261]	1995	Test data generation	Maximise program branch coverage	Input Vector	GA	Synthetic
Roper *et al.* [401]	1995	Test data generation	Maximise program coverage	Input Vector	GA	Synthetic
Sthamer [445]	1995	Test data generation	Maximise Branch Coverage	Input Vector	GA	?
Watkins [478] N	1995	Test data generation	?	Input Vector	GA	?
Alander *et al.* [13]	1996	Test case generation	Minimise the response time	?	GA	Real
Ferguson and Korel [178]	1996	Test case generation	Improve hard-to-cover predicate branches coverage	Input Vector	Local Search (Chaining Approach)	Real
Jones *et al.* [262]	1996	Test data generation	Maximise branch coverage	Input Vector	GA	Real
Roper [399] N	1996	Test data generation	Maximise code coverage	Input Vector	GA	?
Alander *et al.* [15]	1997	Testing Software Response Times	Minimise the response time	?	GA	?
Alander *et al.* [14]	1997	Test case generation	Minimise the response time	?	GA	Synthetic

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Baradhi and Mansour [49]	1997	Comparative study of regression testing algorithms	Minimise execution time and the number of test cases, Maximise the Precision	-	GA, SA	? (20 programs)
Michael and McGraw [350]	1997	Test data generation	Maximise condition/decision coverage	Input Vector	SA, GA	Real
Michael *et al.* [351]	1997	Test data generation	Maximise CDC coverage	Input Vector	GA	Synthetic
Roper [400] N	1997	Test data generation	Maximise branch coverage	Input Vector	GA	?
Wegener *et al.* [483]	1997	Execution time determine	Minmise execution time	String	GA	Real
Wegener *et al.* [482]	1997	Test case generation for real-time systems	Optimise worst/best execution time	String	GA	Real
Alander *et al.* [16]	1998	Test data generation for response time extremes	Maximise response time	Input Vector	GA	Synthetic
Borgelt [70] N	1998	Test data generation	Maximise Inverse Path Probability and Last Block Traversal Probability with Bonus	Input Vector	GA	Real (referenced)
Feldt [173] N	1998	Software Diversity	minimise the probability of coincident failures for multi-version systems	Tree	GP	?
Feldt [170] N	1998	Software Diversity	minimise the probability of coincident failures for multi-version systems	Tree	GP	?
Jones *et al.* [263]	1998	Branch and fault-based testing	Maximise branch coverage and fault detection	String	GA	Synthetic?
McGraw *et al.* [339]	1998	Test data generation	Maximise program coverage	Input Vector	GA	Real
Michael and McGraw [349]	1998	Test data generation	Maximise code coverage	Input Vector	SA, GA	Synthetic
Michael *et al.* [352]	1998	Test data generation	Maximise condition/decision coverage	Input Vector	GA, SA	Real
Tracey *et al.* [452]	1998	Test case generation	Optimise worst-case execution time, specification conformance, structural coverage and exception condition testing	?	SA	Real (small cases)
Tracey *et al.* [451]	1998	Test case generation	Optimise: specification failure, exception condition	Input Vector	SA	Real
Wegener and Grochtmann [480]	1998	Test case generation for real-time systems	Optimise worst/best case execution time	Input Vector	EA	Real
Alander and Mantere [11] N	1999	Automatic software testing	?	?	GA	?
Mansour and El-Fakih [329]	1999	Test case selection	Minimise number of test cases needed	Binary string	SA, GA, BB	Synthetic
Moghadampour [365] N	1999	?	?	?	GA	?
Pargas *et al.* [377]	1999	Test data generation	Maximise statement and branch coverage	Input Vector	GA	Real (6 programs)
Pohlheim and Wegener [381]	1999	Verifying worst/best case execution time	Optimise worst/best case execution time	Input Vector	EA	Real
Alander and Mantere [12] N	2000	Automatic software testing	?	?	GA	?
Bueno and Jino [86]	2000	Identification of likely infeasible program paths	Maximise the number of predicates traversed on a program path and minimize the data error on the deviation predicates	Binary string	GA	Real (numerical programs)

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Groß [201] N	2000	Test case generation	Minimise or maximise the execution time	Input Vector	?	?
Groß *et al.* [205]	2000	Predicting accuracy of ET for timing constraint of complex objects	Predict accuracy of ET using a new complexity measure	Input Vector	EA	Real (Sample modules from different applications)
Lin and Yeh [306]	2000	Test case generation	Maximise the target path coverage	Bit string	GA	Synthetic
MacNish [320]	2000	Uncovering logical errors in student code	Maximise logical error detection	String	GA	Real
Tracey *et al.* [453]	2000	Test data generation	Maximise Exception Conditions	Input Vector	GA	Real
Tracey [455]	2000	Test data generation	Optimise metrics concerning: specification falsification, structural, exception condition and worst-case execution time testing	Input Vector	SA, GA	Real
Bueno and Jino [87]	2001	Test data generation - execute program paths	Maximise the number of predicates traversed on a program path and minimize the data error on the deviation predicates	Binary string	GA	Real (numerical programs)
Groß [202]	2001	Timing analysis of real-time systems	Maximise violation of timing constraints	Input Vector	GA	Real (test objects from different applications)
Hanh *et al.* [212]	2001	Comparison of integration testing strategies	Optimising testing resources allocation and minimise the stubs	?	GA	Real
Khurshid [277]	2001	Test data generation	Maximise code coverage with bonus and barrier function	Input Vector	GA	Real (Java INS)
Lin and Yeh [307]	2001	Test data generation	Maximise the selected branch coverage	Input Vector	GA	Synthetic
Mansour *et al.* [331]	2001	Test case selection	Minimise number of selected test cases and execution time, Maximise Precision	Binary string	SA	benchmark programs
Mantere and Alander [333] N	2001	Automatic testing	?	?	GA	?
Michael *et al.* [353]	2001	Test data generation	Optimise test adequacy criterion	Input Vector	GA	Real
Sthamer *et al.* [444]	2001	Test data generation in embedded systems	Maximise the selected structural coverage	Input Vector	GA	Real
Wegener *et al.* [484]	2001	Test case generation for structural testing	Maximise branch coverage	Input Vector	EA (using MATLAB)	Real
Wegener and Mueller [481]	2001	Test data generation for real time systems	Optimise worst/best execution time	Input Vector	EA	Real
Baresel *et al.* [51]	2002	Test data generation	Maximise coverage	Input Vector	EA	Real
Baudry *et al.* [59]	2002	Test data generation for mutation-based testing	Maximise the mutation score	Input Vector	GA (Bacteriological Algorithms)	Real

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Baudry *et al.* [58]	2002	Test data generation for mutation-based testing	Maximise the mutation score	Input Vector	GA (Bacteriological Algorithms)	Real
Bottaci [72]	2002	Boolean flag problem	tackle the special and general cases	Input Vector	GA	-
Briand *et al.* [79]	2002	Integration test case orders optimisation	Minimise the complexity of stubbing	String	GA	?
Bueno and Jino [88]	2002	Test data generation - execute program paths and Identification of Likely Infeasible program paths	Maximize the number of predicates traversed on a program path and minimize the data error on the deviation predicates	Binary string	GA	Real (numerical programs)
Cornford *et al.* [126]	2002	Defect detection and prevention	Risk reduction	?	GA	Real
Emer and Vergilio [157]	2002	Test data sets selection and evaluation	Maximise fault detection capability	Tree	GP	Real
Groß and Mayer [203]	2002	Execution time analysis of component based real-time systems	Find optimal task input parameter combinations within timing constraint	Input Vector	EA	Real
Harman *et al.* [226]	2002	Improving searchability by flag removal	Maximise coverage	Input vector	EA	Real (simple)
Harman *et al.* [225]	2002	Test data generation	Maximise test data adequacy criterion	Input Vector	EA	-
Mansour and Bahsoon [328]	2002	Test case selection	Maximise test code coverage	Input Vector	SA	?
Tracey *et al.* [454]	2002	Test data generation	Optimise metrics concerning: specification falsification, structural, exception condition and worst-case execution time testing	Input Vector	GA	Real
Wegener *et al.* [486]	2002	Test case design	Improve quality of evolutionary testing in embedded systems	Input Vector	GA	Synthetic
Wegener *et al.* [485] N	2002	Test data generation	?	Input Vector	?	?
Baresel *et al.* [52]	2003	Automatic sequence testing	Maximise structural coverage	String	EA	Real
Baresel and Sthamer [50]	2003	Test data generation	Maximise coverage	Input Vector	EA	Real
Berndt *et al.* [64]	2003	Test case generation	Maximise Novelty, Minimise proximity	Input Vector	GA	benchmark program
Bottaci [73]	2003	Defining different cost functions for test data generation	Covering specific branches	Input Vector	EA	Synthetic
Cohen *et al.* [121]	2003	Test suites generation	Maximise code coverage	Input Vector	SA	Synthetic
Cohen *et al.* [122]	2003	Interaction Testing	Maximise pair-wise and t-way coverage	Arrays	HC, SA	Synthetic
Cornford *et al.* [127]	2003	Defect detection and prevention	Risk reduction (Minimise risk)	Boolean vector	GA, SA	Real
Díaz *et al.* [145]	2003	Test case generation	Maximise branch coverage	Input Vector	TS	Synthetic
Emer and Vergilio [158]	2003	Test data selection and evaluation	Maximise code coverage	Tree	GP	Synthetic
Ghazi and Ahmed [191]	2003	Interaction testing	Maximise pairwise test coverage	Array	GA	Synthetic
Groß and Mayer [204]	2003	Component testing of real-time systems	Maximise worst-case execution time	-	GA	-

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Guo *et al.* [207]	2003	Conformance testing of UIO sequences for FSMs	Maximise the number of discrete units; Minimise the length of sequence	String	GA	Synthetic
Hermadi and Ahmed [240]	2003	Test data generation	Maximise path coverage	Input Vector	GA	Synthetic
Korel *et al.* [282]	2003	Test data generation	Cover desired part of code	Input Vector	Road-Map Based Search	Real
Li and Lam [298]	2003	State-based Test Suites	Optimisation of State-based Test Suites	String	Goal-Oriented Evolution Strategy (GOES)	Benchmark example
Mantere [332]	2003	Automatic software testing	-	-	GA	-
May *et al.* [336]	2003	Mutation testing	Fault detection	Bit string	AIS	-
McMinn and Holcombe [341]	2003	State problem for ET	Maximise coverage (by generating sequence of inputs)	Graph	ACO	Small example
Patton *et al.* [378]	2003	Software usage testing	Maximise likelihood of occurring, Maximise failure intensity	String	GA	Synthetic
Sagarna and Lozano [404]	2003	Test data generation	Maximise coverage	Input Vector	EDA	From literature
Adamopoulos *et al.* [1]	2004	Test data and mutant generation	Minimise test set	Input Vector	GA	Synthetic
Baresel *et al.* [54]	2004	Test data generation	Maximise coverage	Input Vector	GA	Real
Baresel *et al.* [53]	2004	Test case generation	detect critical defects	Input Vector	EA	?
Berndt and Watkins [62]	2004	Test suite generation	Maximise Novelty, Minimise proximity	Input Vector	GA	Real
Chan *et al.* [99]	2004	Action sequences in computer games	Detect and identify unwanted behavior	String	GA	Real
Ferreira and Vergilio [179]	2004	Test data generation	Maximise code coverage	Input Vector	GA & Hybrid GA	?
Harman *et al.* [227]	2004	Testability transformation	Improve evolutionary test data generation in presence of flag variables	Input Vector	EA	4 small programs
Hermadi [239]	2004	Test data generation	Maximise program coverage	Input Vector	GA	-
Hierons *et al.* [241]	2004	Input sequence generation for CFSMs	Minimise the cost of reaching the transition	String	GA	-
Lammermann *et al.* [289]	2004	Test case design	Maximise code coverage	?	?	Real
Li and Wu [301] N	2004	Software test automation	Minimal multiple-condition coverage test	Input Vector	EAs	Real
Mansour and Salame [330]	2004	Test case generation	Maximise path coverage	Input Vector (Integer and real value)	SA, GA	benchmark programs
McMinn and Holcombe [342]	2004	Test data generation	Maximise branch coverage	Input Vector	Hybrid EA (with chaining approach)	From literature
Tonella [450]	2004	Test case generation	Maximise given coverage, Minimise test set	Input Vector	GA	Real
Wappler [468] N	2004	Test of OO system	?	?	EAs	?

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Watkins *et al.* [476]	2004	Test case generation in system testing	Maximise the targeted coverage	Input Vector	GA	Synthetic
Wegener and Bühler [479]	2004	Test case generation	Maximise functional errors using two fitness function based on distance and area criteria	Input Vector	EA	Real
Zhan and Clark [500]	2004	Test data generation	Maximise structural coverage	Input Vector	SA	Synthetic
Zhang [506] N	2004	Test data generation	?	Input Vector	GA	?
Alba and Chicano [18]	2005	Test data generation	Maximise condition-decision coverage	Input Vector	ES, GA	11 benchmark test programs
Baudry *et al.* [60]	2005	Test data generation for mutation-based testing	Maximise the mutation score	Input Vector	GA (Bacteriological Algorithms)	Real
Baudry *et al.* [61]	2005	Test data generation for mutation-based testing	Maximise the mutation score	Input Vector	GA (Bacteriological Algorithms)	Real
Berndt and Watkins [63]	2005	Test suite generation	Maximise Novelty, Minimise proximity	Input Vector	GA	Real
Boström and Björkqvist [71]	2005	Test case generation	Maximise the number of assertion get	Input Vector	SQP	Synthenic
Briand *et al.* [80]	2005	-	Maximise the chances of critical deadline misses within the system	String (of pairs)	GA	Synthetic & Real
Bryce and Colbourn [82]	2005	Test suite construction for interaction testing	Covering all t-tuples with minimum test suite	Array	GS	Synthetic
Bryce *et al.* [85]	2005	Test suite construction for interaction testing	Minimise size of the covering test suite for pair-wise coverage	Array	GS	Real
Del Grosso *et al.* [133]	2005	Stress test data generation	Minimise Buffer Overflow threat	Input Vector array	GA	Real (open source)
Derderian *et al.* [136]	2005	Input sequences and transition paths generation for EFSM	Optimise the generation of feasible transition paths (FTPs)	?	GA	Real
Dillon [147]	2005	Determination of worst case execution time of embedded systems	run-time of the test data exercised on the code	Input Vector	HC	benchmark programs
Girgis [193]	2005	Test data generation	Maximise path-coverage	Input Vector	GA	Real (small?)
Guo *et al.* [208]	2005	Unique I/O sequence generation for FSMs	Maximise number of discrete units and Minimise length of the sequence	String	GA, SA	Small example (FSMs)
Hierons *et al.* [242]	2005	Program restructuring transformation	Maximise branch coverage	Input Vector	EA (implicitly using testability transformation)	examples
Korel *et al.* [283]	2005	Testability transformation	Maximise branch coverage	Input Vector	HC (AVM)	Synthetic
Lammermann and Wappler [288]	2005	Test effort estimation	-	-	EA	-

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Li and Lam [300]	2005	Test threads generation from UML diagrams	improve the quality of test threads generation	Graph	improved ACO	Real
Li and Lam [299]	2005	Automatic test sequence generation	Maximise all-state coverage and feasibility	Graph	ACO	Benchmark example
Liu *et al.* [309]	2005	Test data generation	fitness function development	Input Vector	EA	Real?
Liu *et al.* [310]	2005	Test data generation	Maximise code coverage	Input Vector	Ant PathFinder Algorithm (ACO, GA)	Real (open source)
Masud *et al.* [335]	2005	Test data generation in mutation testing	Fault detection	Input Vector	GA	-
McMinn *et al.* [345]	2005	Testability transformation for test data generation	Maximise coverage	Input Vector	EA	Real (two cases)
McMinn and Holcombe [343]	2005	Test data generation	Maximise branch coverage	Input Vector	Hybrid EA (with chaining approach)	Real and Literature
Regehr [391]	2005	Interrupt-driven software testing	Improve the quality of interrupt schedule	List of vectors	GA	Real
Sagarna and Lozano [405]	2005	Test data generation	Maximise branch coverage	Input Vector	EDA	Real
Tsai *et al.* [456]	2005	Clustering for WS group testing	Find the cluster of the correct WS outputs	Input Vector	SA	Synthetic & Real
Wappler and Lammermann [470]	2005	Test case generation for the unit testing of OO software	Maximise the statement,branch and condition coverage	Input Vector	EA	Real (simple cases)
Xie *et al.* [497]	2005	Test case generation	Maximise code coverage	Input Vector	GA, SA	-
Xie *et al.* [496]	2005	Test case generation	Maximise statement/branch coverage	Input Vector	GA	Synthetic
Zhan and Clark [501]	2005	Test data generation for MATLAB Simulink models	Maximise test adequacy in mutation testing	Input Vector	SA	Real?
Alshraideh and Bottaci [26]	2006	Test data generation	Maximise program branch coverage	Input Vector	GA	Real
Alshraideh and Bottaci [27]	2006	Test data generation	Maximise code coverage	Input Vector	GA	Real
Briand *et al.* [81]	2006	Schedulability of test cases in real-time system	Executing triggered tasks within time constraints	Matrix	GA	Synthetic
Cheon and Kim [110]	2006	Evolutionary testing for OO programs	?	?	EA	Synthetic
Derderian [138]	2006	Test sequence generation for Finite State Machines	generate feasible transition paths (FTPs)	Binary string	GA	benchmark FSM
Derderian *et al.* [137]	2006	Generating UIO sequences for FSM conformance testing	Maximise FSM non-conformance?	String	GA	benchmark
Everson and Fieldsend [161]	2006	Tuning parameters of a safety related system	Minimise false positive%, Maximise true positive% Minimise mean additional warning time	String	MOES	Real
Garousi [186]	2006	Test data generation in stress testing	Maximise the traffic on a specified network or node	Input Vector	GA	Real
Garousi *et al.* [188]	2006	Stress testing in real-time systems	Maximise the traffic on a specified network or node	Input Vector	GA	Real
Guo [206]	2006	Fault identification in FSM testing	Maximise discrete units and Minimise the solution length (Minimise the cost of fault identification)	String	GA, SA	Synthetic

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Mahanti and Banerjee [321]	2006	Model checking	Maximise error detection	Graph?	ACO, Self Regulating Swarm Agents	Synthetic
Mayer [338]	2006	Test data generation	Maximise failure rate	Input Vector	Random Search	-
McMinn *et al.* [346]	2006	Test data generation	Maximise branch coverage (via transformation)	Input Vector	EA	Real (two cases)
McMinn and Holcombe [344]	2006	Test data generation	Find test data missed	tree	hybrid approach	Real
Miller *et al.* [354]	2006	Test data generation	Maximise condition-decision coverage	Enumerated	GA	Synthetic
Sagarna and Lozano [406]	2006	Test data generation	Maximise branch coverage (of C/C++ programmes	Input Vector	SS, EDA	From Literature
Seesing [418]	2006	Test data generation	Maximise branch coverage	Tree	GP	Real
Seesing and Groß [419]	2006	Test data generation	Maximise branch coverage	Tree	GP	Real
Seesing and Groß [420]	2006	Test data generation	Maximise branch coverage	Tree	GP	Real
Seesing and Groß [421]	2006	Test data generation	Maximise branch coverage	Tree	GP	Real
Shan and Zhu [424]	2006	Test data generation	Maximise fault detection capability	Input Vector	Data Mutation	Real
Tlili *et al.* [449]	2006	Test data generation	Optimise worst/best case execution time	Input Vector	EA	Real
Uyar *et al.* [457]	2006	Structural test data generation	Maximise matching pattern	Input Vector	EA	Simple cases
Walcott *et al.* [464]	2006	Regression test suite prioritization	Maximise code coverage	String	GA	Real
Wang [465]	2006	Test data generation	Maximise branch coverage	Input Vector	GA	Real
Wappler and Wegener [473]	2006	Test case generation for unit testing of OO software	Maximise branch coverage using three distance metrics	Tree	GP	Real (from java standard classes)
Wappler and Wegener [472]	2006	Test cases generation for object-oriented software	Maximise branch coverage	tree	hybrid GP & GA	Synthetic
Watkins and Hufnagel [475] N	2006	Test data generation	fitness functions comparison	Input Vector	GA	?
Watkins *et al.* [477] N	2006	Test suites generation	Detect the system failures	Input Vector	GA	?
Xiao *et al.* [494]	2006	Test sequences generation in protocol conformance testing	Generation of the shortest possible test sequence	Graph	-	Synthetic
Zhan and Clark [502]	2006	Test data generation for MATLAB Simulink models	Maximise coverage	Input Vector	SA	Real
Andreou *et al.* [30]	2007	Test data generation	Maximise code coverage of generated data flow paths	Input Vector	GA	Synthetic
Arcuri and Yao [40]	2007	Test data generation for Java Containers	Maximise code coverage; Minimise input sequence length	Input Vector	HC, SA, GA	Real
Arcuri and Yao [37]	2007	Test data generation for container classes	Maximise the coverage; Minimise the branch distance and the length	Input Vector	HC, GA, MA	Real

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Arcuri and Yao [39]	2007	Test data generation	Maximise the coverage	-	-	-
Arcuri and Yao [38]	2007	Coevolve programs and unit tests	Maximise the ability to pass the unit tests	-	GP	Real (open source)
Ayari *et al.* [46]	2007	Test input data generation for mutation testing	Maximise the mutation score (MScore)	Input Vector	RS, HC, GA, ACO	Real (benchmark: triangle)
Blanco *et al.* [67]	2007	Test suite generation	Maximise branch coverage	Input Vector	SS	Triangle benchmark
Bryce and Colbourn [83]	2007	Interaction Test suite selection	Maximise t-way coverage	Array	GS, HC, SA, TS	Synthetic
Bryce and Colbourn [84]	2007	Interaction testing	Minimise the size of test suites and execution time for pairwise coverage	Array	GS	Synthetic
Bueno *et al.* [89]	2007	Improving Random Test Sets Using the Diversity Oriented Test Data Generation	Optimize the diversity among the test data on a test data set	Input vector	GA, SA, Simulated Repulsion	Real (numerical programs)
Cohen *et al.* [123]	2007	Combinatorial interaction testing	produce the smallest subset of configurations for t-way coverage	Array	SA	Synthetic
de Abreu *et al.* [132]	2007	Test data generation for programs having paths with loops	Maximise path coverage	Input Vector	Generalized Extermal Optimisation (GEO)	7 benchmark programs
Di Penta *et al.* [143]	2007	Test inputs generation and configurations	Maximise code coverage	Input Vector	GA, Random Search	Real
Ghiduk *et al.* [192]	2007	Test data generation	Maximise data-flow coverage	Input Vector	GA	Synthetic & Real
Guo *et al.* [209]	2007	Fault diagnosis in Finite State Machines Testing	Minimise the conflict set; Isolate and identify the faults	String	U-method	Synthetic
Harman *et al.* [231]	2007	Test data generation	Theoretical and empirical support for ET	Input Vector	HC, GA	Real
Harman *et al.* [230]	2007	Test data generation	Maximise branch coverage and dynamic memory allocation	Input Vector	MOOA (NSGA-II)	Synthetic
Harman and McMinn [221]	2007	Structural test data generation	Theoretical and empirical analysis of ET	Input Vector	HC	Real
ming Hsu [356] N	2007	Test data generation for program changes	Maximise program coverage, Minimise cost	Input Vector	GA	?
Khamis *et al.* [271] N	2007	Test data generation	Maximise spanning sets coverage	Input Vector	GA	?
Lefticaru and Ipate [292]	2007	Test data generation for state-based testing	Maximise the closeness to a given path	Input Vector	GA	Synthetic
Lehre and Yao [295]	2007	Run-time analysis of (1+1)ES in finding UIO sequences for FSMs	Minimise the length of UIOs	String	ES	Synthetic
Levin and Yehudai [297]	2007	Test data generation	Maximise branch coverage	Input Vector	GA	Synthetic
Li *et al.* [302]	2007	Regression test case prioritization	Maximise block coverage	String?	HC, GA	Real?
Liaskos and Roper [303]	2007	Test data generation	Maximise data-flow (d-u) coverage	Input Vector	GA, AIS	

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Liaskos *et al.* [305]	2007	Test data generation for OO software	Maximise data-flow (d-u) coverage	Input Vector	GA	Real (standard Java library)
Liu *et al.* [311]	2007	Fitness calculation in evolutionary testing	improve the fitness calculation quality	-	-	-
Liu *et al.* [312]	2007	Test data generation and evaluation	Optimise fitness function	Input Vector	GA	Synthetic
May *et al.* [337]	2007	Mutation testing	Minimise the time taken to generate test set, Maximise mutation score	String	GA, AIS	Synthetic
Ribeiro *et al.* [396]	2007	Test data generation and optimisation	Maximise code coverage	Tree	GP	-
Ribeiro *et al.* [394] N	2007	Test data generation	Maximise code coverage	Tree	GP	?
Ribeiro *et al.* [395] N	2007	Test data generation	Maximise structural code coverage	Input Vector	?	
Sagarna [403]	2007	Test data generation	Maximise branch coverage	Input Vector	EDA, SS	Real
Sagarna *et al.* [410]	2007	Test data generation for containers in the context of OO software	Maximise branch coverage	Input Vector	EDA	Real
Sagarna and Lozano [407]	2007	Test data generation to fulfill branch coverage	Measure suitability of EDA	Input Vector	Data Mining, EDA	Synthetic
Sofokleous and Andreou [440]	2007	Test case generation	Maximise selected coverage	Input Vector	Hybrid GA	Synthetic & Real
Waeselynck *et al.* [463]	2007	Robustness testing of cyclic real-time control system	Search for dangerous scenarios and yield a violation of system-level safety requirements	Input Vector	SA	Real?
Wappler *et al.* [474]	2007	Code transformation	Maximise code coverage	Input Vector	GA	Real
Wappler and Schieferdecker [471]	2007	Generation of OO unit tests	Maximise structural coverage	EAs	Input Vector	
Windisch *et al.* [492]	2007	Test data generation	Maximise structural code coverage	Input Vector	GA, PSO	Synthetic & Real
Xiao *et al.* [495]	2007	Goal-oriented test data generation	Maximise decision-coverage	Input Vector	GA, SA	Real (5 programs)
Yoo and Harman [499]	2007	Test case selection	Maximise coverage, Minimise execution time, Minimise cost	String	GS, MOOA (NSGA-II)	Real (small?)
Ahmed and Hermadi [9]	2008	Test data generation	Maximise path coverage	Input Vector	GA	Synthetic
Alba and Chicano [23]	2008	Test data generation	Maximise coverage	Input Vector	GA, ES	Real (12 test problems)
Arcuri [36]	2008	Automatic bug fixing	distance function	Tree	GP	Synthetic
Arcuri *et al.* [44]	2008	Test data generation	Minimise runtime	Input Vector	Random Search, HC, Alternating Variable Method	Real? (Triangle Classification Problem)

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Arcuri and Yao [41]	2008	Automatic bug fixing	distance function	Tree	GP	Synthetic
Arcuri and Yao [42]	2008	Unit tests generation for container classes	Maximise coverage and Minimise the length of the test sequences	Input Vector	RS, HC, SA, GA, Memetic Algorithms	Real
Bueno *et al.* [90]	2008	Automatic Test Data Generation Using Particle Systems	Optimize the diversity among the test data on a test data set	Input vector	GA, SA, Simulated Repulsion	Synthetic (simulation)
Bühler and Wegener [91]	2008	Functional testing	automatic finding faults	-	-	Real
Chen and Zhong [108]	2008	Test data generation	Maximise target path coverage	Input Vector	GA	Benchmark
Cohen *et al.* [124]	2008	Combinatorial interaction testing	generate the smallest subset of configuration for t-way coverage	Array	Greedy	Real
Del Grosso *et al.* [134]	2008	Buffer overflows detection	Maximise (vulnerable) statement coverage	String	GA	Real
Díaz *et al.* [146]	2008	Test case generation	Optimise cost function	Graph	Tabu Search	Synthetic
Feldt *et al.* [176]	2008	Fitness function evaluation	Search for universal test diversity metrics	-	-	Real (open source)
Garousi [187]	2008	Test data generation	meet evaluation criteria	Input Vector	GA	Real
Garousi *et al.* [189]	2008	Stress testing in real-time systems	Increasing chances of discovering faults	Binary String	GA	Real
Ghani and Clark [190]	2008	Specification inference and test data generation	improve the accuracy of inferred specifications	Input Vector	SA	Real (open source)
Gotlieb *et al.* [199]	2008	Test data generation	-	Input Vector	HC, SA	-
Gupta and Rohil [210]	2008	Test data generation	Maximise program coverage	Input Vector	GA	Real (open source)
Harman [216]	2008	Testability Transformation	Maximise branch coverage	Input Vector	EA	-
Hla *et al.* [243]	2008	Test case prioritization	Maximise the statement, branch and functional coverage	Input Vector	PSO	Real
Jia and Harman [253]	2008	Mutant generation	Search for subsuming Higher Order Mutant	Input Vector	GA, HC, GS	Real (open source)
Jia and Harman [254]	2008	Higher-Order mutation testing	Generating High-quality mutants	String	GA, HC, GS	Real
Kalaji *et al.* [265]	2008	Test data generation	Maximise transition coverage in EFSM	Input Vector	ET-based	Synthetic
Lakhotia *et al.* [286]	2008	Test data generation (Dynamic data structures in testing)	Maximise code coverage	Input Vector	HC	Synthetic & Real
Lefticaru and Ipate [293]	2008	Test data generation	Maximise target path coverage	Input Vector	SA, GA, PSO	Real
Lefticaru and Ipate [294]	2008	Measurement of fitness function in testing	Optimise general form of fitness function	-	SA	-
Lehre and Yao [296]	2008	Construction of UIO sequences for FSMs	Minimise the length of input sequence	String	EAs	Synthetic
Liaskos and Roper [304]	2008	Test data generation	Maximise dataflow coverage	Input Vector	GA	Real
Makai [325] N	2008	Test case generation	Maximise branch and statement coverage	?	Input Vector GA	?

Continued on next page

TABLE VII. Papers on Testing and Debugging – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
McMinn *et al.* [347] N	2008	Test data generation	Maximise branch coverage	Input Vector	GA, HC	Real
Nguyen *et al.* [369]	2008	Autonomous Distributed Systems (ADS) testing	Improve the process of test cases generation	Input Vector	random & evolutionary-mutation	Real
Prutkina and Windisch [382]	2008	Test data generation	Maximise structural coverage	Input Vector	EA	-
Rajappa *et al.* [388]	2008	Test case generation	Maximise code coverage	Input Vector	GA	Synthetic
Ribeiro [393]	2008	Test case generation	Maximise structural coverage	Tree	GP	Real
Ribeiro *et al.* [398]	2008b	Test case generation	Maximise code coverage	Tree	GP	Real
Ribeiro *et al.* [397]	2008a	Test case generation	Maximise code coverage	Tree	GP	Real
Sagarna and Lozano [408]	2008	Test data generation	Maximise code coverage	Input Vector	EDA	Real
Sagarna and Yao [409]	2008	Test data generation	Maximise code coverage and Minimise the constraints (penalty)	Input Vector	single and multi-objective GA	Real (open source)
Schneckenburger and Schweiggert [413]	2008	Test data generation	P-measure (probability of a given test set to detect a failure) and E-measure (average number of failures detected by the test set)	Input Vector	HC	Real
Sofokleous and Andreou [441]	2008	Test data generation	Maximise edge/condition coverage	Input Vector	GA	Synthetic
Sofokleous and Andreou [442] N	2008	Test data generation	Maximise code coverage of generated data flow paths	Input Vector	GA	Synthetic
Wang *et al.* [466]	2008	Fitness function design	Maximise branch coverage	Input Vector	EA	Real (Open Source)
Wappler [469]	2008	Generation of OO unit tests	Maximise code coverage	Tree	GP, GA	Real
Windisch [491]	2008	Introduction of evolutionary structural testing in system models	-	-	EA	-
Zhan and Clark [503]	2008	Test data generation and selection	Maximise branch coverage, Minimise time cost	Input Vector	SA	Synthetic
Zhong *et al.* [510]	2008	Test suite reduction	Minimise the number of selected test cases	Bit string	GA, IP	Real

TABLE VIII. Papers addressing Distribution, Maintenance and Enhancement activities

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Mancoridis *et al.* [326]	1998	Software structure clustering	Maximise inter-connectivity (High cohesion); Minimise intra-connectivity (Low coupling)	Array of module identifiers	HC, GA	Synthetic
Nisbet [370]	1998	Compiler optimisation via determining program transformation sequence	Minimise execution time	String	GA	Synthetic
Williams [490] N	1998	Automatic parallelization	?	?	EAs	?
Cooper *et al.* [125]	1999	Compiler optimisation	Find customized compiler optimisation sequences; Minimise the size of the object code	String	GA	Benchmark programs in FORTRAN and C
Doval *et al.* [152]	1999	Software module clustering	Maximise inter-connectivity (High cohesion); Minimise intra-connectivity (Low coupling)	Array of module identifiers	GA	Real
Mancoridis *et al.* [327]	1999	Automatic clustering of system structure	Maximise inter-connectivity (High cohesion); Minimise intra-connectivity (Low coupling)	Array of module identifiers	GA	Synthetic
Ryan [402] N	2000	Automatic re-engineering of software	?	?	GP	Real
Harman *et al.* [224]	2002	Software module clustering	Optimise granularity, cohesion and coupling metrics	Array of module identifiers	GA	Synthetic
Mitchell [358]	2002	Software structure clustering	Maximise inter-connectivity (High cohesion); Minimise intra-connectivity (Low coupling)	Array of module identifiers	HC, GA, Exhaustive Search	Real
Mitchell and Mancoridis [359]	2002	Software module clustering via improved Hill Climbing	Optimise MQ metric	Array of module identifiers	HC	Real
Mitchell *et al.* [363]	2002	Reverse Engineering from source code	Extract Design Structure	Array of module identifiers	HC, GA, Exhaustive Search	Real (open source)
Sahraoui *et al.* [411] N	2002	Object identification in legacy code	Minimise coupling and Maximise cohesion	Array of sets of data representing candidate objects	GA	Real
Van Belle and Ackley [458]	2002	Code factoring	Performance on a randomly changing symbolic regression problem	Tree	GP	Synthetic
Antoniol *et al.* [32]	2003	Libraries refactoring	Minimise the Dependency Factor (DF), Linking Factor (LF) and Standard Deviation Factor (SF)	Bit-matrix	GA	Real
Fatiregun *et al.* [163]	2003	Program transformation	Find the optimum sequence of transformation rules	Sequence of transformation identifiers	HC	-
Mahdavi *et al.* [324]	2003	Software module clustering	Maximise the number of internal edges; Minimise the number of external edges	Array of module identifiers	HC, GA, SA	Real
Mahdavi *et al.* [323]	2003	Software module clustering	Maximise the number of internal edges; Minimise the number of external edges	Array of module identifiers	HC	Real

Continued on next page

TABLE VIII. Papers on Distribution, Maintenance and Enhancement – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Mitchell and Mancoridis [360]	2003	Software Clustering	Maximise inter-connectivity (High cohesion); Minimise intra-connectivity (Low coupling)	Array of module identifiers	HC, GA, Exhaustive Search	Real
Reformat *et al.* [389]	2003	Automatic generation of clones	Measure feasibility of using GP for clone generation	Parse tree	GP	Real (open source)
Cowan *et al.* [130] N	2004	Evolutionary programming	?	?	GP	?
Fatiregun *et al.* [164]	2004	Program transformation	Minimise program size	Sequence of transformation identifiers	GA, HC, RS	Synthetic
Mitchell *et al.* [364]	2004	Software module clustering using Bunch tool	Maximise inter-connectivity (High cohesion); Minimise intra-connectivity (Low coupling)	Array of module identifiers	HC, SA	Synthetic
Di Penta [140]	2005	Software system renovation	Tradeoff among Dependency Factor (DF), Partitioning Ratio (PR), Standard Deviation Factor (SDF)	bit-matrix	GA	Real
Di Penta *et al.* [142]	2005	Software system renovation	Tradeoff among Dependency Factor (DF), Partitioning Ratio (PR), Standard Deviation Factor (SDF) and Feedback Factor (FF)	Bit-matrix	GA, HC	Real
Di Penta and Taneja [141]	2005	Automatic grammar evolution	Optimise the percentage of positive examples and negative examples	Array of symbols	GA	Real
Fatiregun *et al.* [165]	2005	Amorphous slicing	Minimise size of the amorphous slice computed	Sequence of transformation identifiers	GA, HC	Real (6 programs)
Harman *et al.* [228]	2005	Software module clustering	Optimise MQ and EVM (comparison of two fitness)	Array of module identifiers	GA	Synthetic & Real
Mahdavi [322]	2005	Comprehension via module clustering	Maximise Modularization Quality (Maximise Cohesion and Minimise Coupling)	Array of module identifiers	GA, HC	Real
Seng *et al.* [422]	2005	System re-structuring	Optimise weighted sum of: Coupling, Cohesion, Complexity, Cycles, Bottlenecks	String	GA	Real
Sutton *et al.* [447]	2005	Clone detection	Minimizing number of individuals and Maximizing similarity of clones in each individual	Variable-sized vectors	EA	Real (small case)
Bate and Emberson [57]	2006	Allocation and scheduling tasks in real-time embedded systems	Improve the flexibility	String	SA	Synthetic
Bouktif *et al.* [76]	2006	Mutation and Coverage testing	Clone refactoring	String (binary)	GA, ACO, TS	Real
Cohen *et al.* [120]	2006	Thread clustering	Maximise modularization quality	Array of module identifiers	HC	benchmark example
Del Rosso [135]	2006	Dynamic memory configuration	Improve memory efficiency (find the optimal configuration for the segregated free lists)	String	GA	Simulator
Gold *et al.* [195]	2006	Overlapping concept assignment	Maximise quality of concept binding	String (a set of segment pairs)	GA, HC	Real
Harman [213]	2006	SBSE for Maintenance and Reengineering	-	-	-	-
Mitchell and Mancoridis [361]	2006	Software module clustering	Maximise Modularization Quality (Maximise Cohesion and Minimise Coupling)	Array of module identifiers	HC, SA	Real

Continued on next page

TABLE VIII. Papers on Distribution, Maintenance and Enhancement – *continued from previous page*

Authors [Ref]	**Year**	**Activity**	**Objective / Fitness**	**Representation method**	**Search technique**	**Test problems**
O'Keeffe and Ó Cinnéide [373]	2006	Refactoring of object-oriented programmes	Maximise design quality metric (three metrics were examined)	?	HC, SA	Real
Seng *et al.* [423]	2006	Refactoring	Optimise weighted sum of: Coupling, Cohesion, Complexity, Stability	String	EA	Real
Bodhuin *et al.* [69]	2007b	Model refactoring	Maximise cohesion and minimise coupling	Bit string	GA	Synthetic
Bodhuin *et al.* [68]	2007a	Re-packaging downloadable applications (Software clustering)	Minimise the packaging size and the average of downloading times	Integer array	GA	Real
Emberson and Bate [156]	2007	Task allocation and scheduling in mode transitions	Minimise changes in allocation	?	SA	Synthetic
Harman and Tratt [222]	2007	Refactoring	Maximise coupling between objects (CBO) and Minimise STDV of the number of methods in classes	Sequence of method moves	HC	Real
Huynh and Cai [249]	2007	Software modularity analysis (Clustering)	check the consistency between design and source code	?	GA	Synthetic
O'Keeffe and Ó Cinnéide [374]	2007	Software refactoring	QMOOD hierarchical design quality model	Binary string	HC, SA, GA	Real (Open source)
Reformat *et al.* [390]	2007	Software Cloning	Produce a system from its external interactions	Tree	GP	Synthetic
Kessentini *et al.* [270]	2008	Model transformation	Maximise completeness and consistency of source model transformation	M-dimensional vector	PSO	Synthetic
Kuperberg *et al.* [285]	2008	Performance prediction	Create platform-independent parametric performance models	Tree	GP	Synthetic
Mitchell and Mancoridis [362]	2008	Software clustering	Improve modularization quality	Sequence of transformation identifiers	SA	-
O'Keeffe and Ó Cinnéide [375]	2008	Software refactoring	QMOOD hierarchical design quality model	Binary string	SA, GA, HC	Real (Open source)
O'Keeffe and Ó Cinnéide [376]	2008	Software refactoring	QMOOD hierarchical design quality model	Binary string	HC, SA	Real (Open source)
White *et al.* [488]	2008	Non-functional properties satisfaction	Tradeoffs between power consumption and functionality	Binary	GP, MOOA (SPEA2)	Synthetic

TABLE IX. Papers addressing activities related to Metrics

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Harman and Clark [217]	2004	Overview of Search-based approaches and metrics to SBSE	Guidelines to define good fitness functions	-	HC, SA, GA	-
Vivanco and Pizzi [461]	2004	Software Metrics classification	Improve the prediction of software object quality	Bit string	GA	Real
Lange and Mancoridis [290]	2007	Metrics classification (Identify software developer based on style metrics)	Maximise the correct classification	String	GA	Real (Open source)
Vivanco and Jin [462]	2007	OO source code metrics selection	LDA classifier	bit-mask	GA	Real
Vivanco and Jin [460]	2008	Software metrics selection	Improve the prediction of software object quality	Bit string	GA	Real

TABLE X. Papers addressing Management activities

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Chao *et al.* [105] N	1993	Software Project Management	?	?	GA	?
Chang [100]	1994	Software Project Management	-	-	GA	-
Chang *et al.* [101]	1994	Software Project Management	Determine the resource allocation	?	GA	-
Chang *et al.* [102]	1998	Project scheduling and resource allocation	Minimise total cost and finishing time	String	GA	Synthetic
Dolado and Fernandez [150]	1998	Software development effort estimation	Optimise effort estimation	Tree	GP, NN	Synthetic & Real
Evett *et al.* [162]	1999	Software Quality Modeling	Predict software quality	Tree	GP	Real
Dolado [148]	2000	Software size estimation	Optimise software size estimation	Tree?	GP, NN	Real? (from literature)
Shukla [432]	2000	Development effort estimation	Maximise precision of effort estimation	String	NN and GA (GA to train NN)	Real
Aguilar-Ruiz *et al.* [7]	2001	Project Management	maximizing classification percentage and coverage of the rules	String	EA	Real
Burgess and Lefley [92]	2001	Software effort estimation	Maximise accuracy of estimation	Tree	GP	Real (from an existing database)
Chang *et al.* [103]	2001	Project scheduling and resource allocation	Minimise duration and cost of project, Maximise quality of product	graph	GA	Synthetic
Dolado [149]	2001	Cost estimation	Optimise cost estimation	Tree	GP	Real
Jarillo *et al.* [252]	2001	Software development effort estimation	Predict the number of defects, estimate the reliability in terms of time and failure	?	GA, GP	Real
Liu and Khoshgoftaar [314]	2001	Software Quality Classification Model	Minimise the cost of misclassification	Tree	GP	Real
Aguilar-Ruiz *et al.* [8]	2002	Software development's effort, time and quality estimation	Maximise accuracy of estimation	Float string	EA	Real? (from literature)
Bouktif *et al.* [74]	2002	Software Quality prediction	Improve correctness	Binary tree	GA	Real
Kirsopp *et al.* [280]	2002	project effort estimation	Optimise effort estimate precision	String	HC, Forward Sequential Selection	Synthetic
Shan *et al.* [425]	2002	Software development effort estimation	Determine metric sets and improve the prediction of development effort	Tree	GP	Real
Bouktif *et al.* [75]	2004	Software quality prediction	Improve correctness	Binary tree	GA	Real
Khoshgoftaar *et al.* [273]	2003	Software Quality Classification	Minimise the cost misclassification; Minimise the size of decision tree	Tree	GP	Real

Continued on next page

TABLE X. Papers on Management – *continued from previous page*

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Lefley and Shepperd [291]	2003	Software development effort estimation	Maximise accuracy of estimation	Tree	GP	Real
Liu and Khoshgoftaar [315]	2003	Software Quality Classification	Minimise the cost of misclassification; Minimise the size of decision tree	Tree	GP	Real
Antoniol *et al.* [33]	2004	Project Planning	Minimise Project Duration	String	GA, Queuing Theory, Simulation	Real
Antoniol *et al.* [34]	2004	Project Planning	Minimise project duration	String (two types)	HC, SA, GA	Real
Liu and Khoshgoftaar [313]	2004	Software Quality Classification	Minimise the cost of misclassification; Minimise the size of decision tree	Tree	GP	Real
Alba and Chicano [17]	2005	Project management of the whole SE activities	Minimise project duration and cost (conflicting objectives)	Bit string	GA	Synthetic
Antoniol *et al.* [35]	2005	Project Planning	Minimise Project Duration	String	GA, HC, SA, Random Search	Real
Lokan [316]	2005	Software effort estimation	MSE, LAD, MRE, MER and Z,	symbolic expression	GP	Real
Alvarez-Valdes *et al.* [28]	2006	Project scheduling	Minimise finish time	String	SSA	Synthetic
Bouktif *et al.* [77]	2006	Quality planning	Maximise the predictive accuracy	String or Matrix?	SA	Real
Sheta [430]	2006	Development effort estimation	Maximise precision of effort prediction	?	GA	Real
Alba and Chicano [22]	2007	Project management of the whole SE activities	Minimise project duration and cost (conflicting objectives)	String	GA	Synthetic
Khoshgoftaar and Liu [272]	2007	Software Quality Classification Model	Minimise the modified expected cost of misclassification, optimise the number of predicted fault-prone modules and minimise the size of the decision tree model	Tree	GP	Real
Barreto *et al.* [56]	2008	Staffing software project	Maximise the value creation for project	-	BB	Synthetic
Chang *et al.* [104]	2008	Software project scheduling	Minimise the input (overload, project costs, and time)	?	GA	Synthetic
Cortellessa *et al.* [128]	2008a	Decision support for software architecture	Minimise cost under delivery time and quality constraints	?	IP (LINGO based)	Synthetic
Hericko *et al.* [238]	2008	Team size optimisation	Define team size with minimal project effort	-	gradient method	Real
Huang *et al.* [248]	2008	Software effort estimation	Minimise the mean magnitude relative error (MMRE)	String	GA	Real
Kapur *et al.* [266]	2008	Staffing for product release	Provide best quality to customers under time constraint	Binary	GA	Synthetic
Khoshgoftaar *et al.* [276]	2008	Software Quality Classification Modeling	Minimise the cost misclassification; Minimise the size of decision tree	Integer or real values string	GA	Real
Wen and Lin [487]	2008	Multistage human resource allocation	Minimise the project duration; Minimise the project cost	Improved Fixed-length Encoding	GA	Synthetic

TABLE XI. Papers addressing Distributed Artificial Intelligence

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Sinclair and Shami [438]	1997	Design and Implementation (evolution of simple software agents)	Maximise the number of accumulated units	String, Tree	GA, GP	Synthetic
Hodjat *et al.* [244]	2004	Adaptive Agent Oriented Software Architecture (AAOSA)	Improve the success rate and quality of the Policies	?	GA	Synthetic
Haas *et al.* [211]	2005	Parameter tuning for multi-agent systems	Improve the quality of composition for software configuration	Integer string	GA	Synthetic

TABLE XII. Papers addressing Security and Protection

Authors [Ref]	Year	Activity	Objective / Fitness	Representation method	Search technique	Test problems
Dozier *et al.* [153]	2004	Hole discovery in intrusion detection systems (IDS)	Minimise failed detections	String	EA, PSO	Real (Simulated)
Dozier *et al.* [154]	2007	Hole discovery in intrusion detection systems (IDS)	Minimise failed detections	String	EA, PSO	Real (Simulated)

TABLE XIII. Papers addressing General aspects of SBSE

Authors [Ref]	**Year**	**Content**
Clarke *et al.* [117] N	2000	Describing several applications of metaheuristic search techniques in Software Engineering
Harman and Jones [218]	2001	Introduction of SBSE
Harman and Jones [219]	2001	SEMINAL: Software Engineering using Metaheuristic INovative Algorithms
Harman and Jones [220]	2001	Outlining the papers presented at the SEMINAL Workshop and the discussions
Pedrycz [379]	2002	Application of Computational Intelligence in different stages of SE
Clark [114]	2003	Cryptography using nature-inspired search techniques
Clarke *et al.* [118]	2003	Reformulating SE as a search problem
Harman and Wegener [223]	2004	On application of search techniques in Software Engineering
McMinn [340]	2004	Survey of Search-Based Test Data Generation
Rela [392]	2004	A review of EC in all SE activities
Mantere and Alander [334]	2005	A review of Evolutionary Software Engineering
Jiang [255]	2006	A review of applying GAs to Software Engineering problems
Harman [215]	2007	Introducing 8 specific application areas
Harman [214]	2007	A introduction of search based Software Engineering for program comprehension
Jiang *et al.* [256]	2007a	A measure to predict the hardness of GAs to the optimisation problem in SE
Afzal *et al.* [4]	2008	A Review of the articles based on non-functional Search-Based Software Testing in 1996-2007
Alander [10]	2008	A bibliography and collection of GA papers applying to testing problems

REFERENCES

[1] Adamopoulos, K., Harman, M., and Hierons, R. M. (2004). How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation using Co-Evolution. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103/2004 of *Lecture Notes in Computer Science*, pages 1338–1349, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[2] Afzal, W. and Torkar, R. (2008a). A Comparative Evaluation of Using Genetic Programming for Predicting Fault Count Data. In *Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA '08)*, pages 407–414, Sliema, Malta. IEEE Computer Society.

[3] Afzal, W. and Torkar, R. (2008b). Suitability of Genetic Programming for Software Reliability Growth Modeling. In *Proceedings of the International Symposium on Computer Science and its Applications (CSA '08)*, pages 114–117, Hobart, Australia. IEEE Computer Society.

[4] Afzal, W., Torkar, R., and Feldt, R. (2008a). A Systematic Mapping Study on Non-Functional Search-Based Software Testing. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE '08)*, pages 488–493, San Francisco, USA. Knowledge Systems Institute Graduate School.

[5] Afzal, W., Torkar, R., and Feldt, R. (2008b). Prediction of Fault Count Data using Genetic Programming. In *Proceedings of the 12th IEEE International Multitopic Conference (INMIC '08)*, pages 349–356, Karachi, Pakistan. IEEE.

[6] Afzal, W., Torkar, R., and Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*. To appear.

[7] Aguilar-Ruiz, J. S., Ramos, I., Santos, J. C. R., and Toro, M. (2001). An Evolutionary Approach to Estimating Software Development Projects. *Information and Software Technology*, **43**(14), 875–882.

[8] Aguilar-Ruiz, J. S., Santos, J. C. R., and Ramos, I. (2002). Natural Evolutionary Coding: An Application to Estimating Software Development Projects. In *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1–8, New York, USA.

[9] Ahmed, M. A. and Hermadi, I. (2008). GA-based Multiple Paths Test Data Generator. *Computers & Operations Research*, **35**(10), 3107–3124.

[10] Alander, J. T. (2008). An Indexed Bibliography of Genetic Algorithms in Testing. Technical Report 94-1-TEST, University of Vaasa, Vaasa, Finland.

[11] Alander, J. T. and Mantere, T. (1999). Automatic Software Testing by Genetic Algorithm Optimization, A Case Study. In *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering (SCASE '99)*, pages 1–9, Limerick, Ireland.

[12] Alander, J. T. and Mantere, T. (2000). Genetic Algorithms in Automatic Software Testing - Analysing a Faulty Bubble Sort Routine. In H. Hytyniemi, editor, *Proceedings of the 9th Finnish Artificial Intelligence Conference*, volume 2, pages 23–32, Espoo, Finland. Finnish Artificial Intelligence Society.

[13] Alander, J. T., Mantere, T., Turunen, P., and Virolainen, J. (1996). GA in Program Testing. In *Proceedings of the 2nd Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, pages 205–210, Vaasa, Finland.

[14] Alander, J. T., Mantere, T., and Turunen, P. (1997a). Genetic Algorithm based Software Testing. In *Proceedings of the 3rd International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA '97)*, pages 325–328, Norwich, UK. Springer-Verlag.

[15] Alander, J. T., Mantere, T., and Moghadampour, G. (1997b). Testing Software Response Times using a Genetic Algorithm. In *Proceedings of the 3rd Nordic Workshop on Genetic Algorithms and their Applications (3NWGA)*, pages 293–298, Helsinki, Finland.

[16] Alander, J. T., Mantere, T., Moghadampour, G., and Matila, J. (1998). Searching Protection Relay Response Time Extremes using Genetic Algorithm-Software Quality by Optimization. *Electric Power Systems Research*, **46**(3), 229–233.

[17] Alba, E. and Chicano, F. (2005a). Management of Software Projects with GAs. In *Proceedings of the 6th Metaheuristics International Conference (MIC '05)*, pages 13–18, Vienna, Austria. Elsevier Science Inc.

[18] Alba, E. and Chicano, F. (2005b). Software Testing with Evolutionary Strategies. In *Proceedings of the 2nd Workshop on Rapid Integration of Software Engineering Techniques (RISE '05)*, volume 3943 of *Lecture Notes in Computer Science*, pages 50–65, Heraklion, Crete, Greece. Springer.

[19] Alba, E. and Chicano, F. (2007a). ACOhg: Dealing with Huge Graphs. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 10–17, London, England. ACM.

[20] Alba, E. and Chicano, F. (2007b). Ant Colony Optimization for Model Checking. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Proceedings of the 11th International Conference on Computer Aided Systems Theory (EUROCAST 2007)*, volume 4739 of *Lecture Notes in Computer Science*, pages 523–530, Las Palmas de Gran Canaria, Spain. Springer.

[21] Alba, E. and Chicano, F. (2007c). Finding Safety Errors with ACO. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1066–1073, London, England. ACM.

[22] Alba, E. and Chicano, F. (2007d). Software Project Management with GAs. *Information Sciences*, **177**(11), 2380–2401.

[23] Alba, E. and Chicano, F. (2008). Observations in using Parallel and Sequential Evolutionary Algorithms for Automatic Software Testing. *Computers & Operations Research*, **35**(10), 3161–3183.

[24] Alba, E. and Troya, J. M. (1996). Genetic Algorithms for Protocol Validation. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN '96)*, pages 870–879, Berlin, Germany. Springer.

[25] Alba, E., Chicano, F., Ferreira, M., and Gómez-Pulido,

J. A. (2008). Finding Deadlocks in Large Concurrent Java Programs using Genetic Algorithms. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1735–1742, Atlanta, GA, USA. ACM.

[26] Alshraideh, M. and Bottaci, L. (2006a). Search-based Software Test Data Generation for String Data using Program-Specific Search Operators. *Software Testing, Verification and Reliability*, **16**(3), 175–203.

[27] Alshraideh, M. and Bottaci, L. (2006b). Using Program Data-State Diversity in Test Data Search. In *Proceedings of the 1st Testing: Academic & Industrial Conference - Practice and Research Techniques (TAICPART '06)*, pages 107–114, Cumberland Lodge, Windsor, UK. IEEE.

[28] Alvarez-Valdes, R., Crespo, E., Tamarit, J. M., and Villa, F. (2006). A Scatter Search Algorithm for Project Scheduling under Partially Renewable Resources. *Journal of Heuristics*, **12**(1-2), 95–113.

[29] Amoui, M., Mirarab, S., Ansari, S., and Lucas, C. (2006). A Genetic Algorithm Approach to Design Evolution using Design Pattern Transformation. *International Journal of Information Technology and Intelligent Computing*, **1**(2), 235–244.

[30] Andreou, A. S., Economides, K. A., and Sofokleous, A. A. (2007). An Automatic Software Test-Data Generation Scheme Based on Data Flow Criteria and Genetic Algorithms. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology (CIT '07)*, pages 867–872, Fukushima, Japan. IEEE Computer Society.

[31] Antoniol, G. and Di Penta, M. (2003). Library Miniaturization using Static and Dynamic Information. In *Proceedings of the 19th International Conference on Software Maintenance (ICSM '03)*, pages 235–244, Amsterdam, Netherlands. IEEE Computer Society.

[32] Antoniol, G., Di Penta, M., and Neteler, M. (2003). Moving to Smaller Libraries via Clustering and Genetic Algorithms. In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR '03)*, pages 307–316, Benevento, Italy. IEEE.

[33] Antoniol, G., Di Penta, M., and Harman, M. (2004a). A Robust Search-based Approach to Project Management in the Presence of Abandonment, Rework, Error and Uncertainty. In *Proceedings of the 10th International Symposium on the Software Metrics (METRICS '04)*, pages 172–183, Chicago, USA. IEEE Computer Society.

[34] Antoniol, G., Di Penta, M., and Harman, M. (2004b). Search-based Techniques for Optimizing Software Project Resource Allocation. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103/2004 of *Lecture Notes in Computer Science*, pages 1425–1426, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[35] Antoniol, G., Di Penta, M., and Harman, M. (2005). Search-based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 240–249, Los Alamitos, California, USA. IEEE Computer Society.

[36] Arcuri, A. (2008). On the Automation of Fixing Software Bugs. In *Proceedings of the Doctoral Symposium of the IEEE International Conference on Software Engineering (ICSE '08)*, pages 1003–1006, Leipzig, Germany. ACM.

[37] Arcuri, A. and Yao, X. (2007a). A Memetic Algorithm for Test Data Generation of Object-Oriented Software. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC)*, pages 2048–2055, Singapore. IEEE.

[38] Arcuri, A. and Yao, X. (2007b). Coevolving Programs and Unit Tests from their Specification. In *Proceedings of the twenty-second IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*, pages 397–400, Atlanta, Georgia, USA. ACM.

[39] Arcuri, A. and Yao, X. (2007c). On Test Data Generation of Object-Oriented Software. In *Testing: Academic and Industrial Conference, Practice and Research Techniques (TAIC PART)*, pages 72–76, Cumberland Lodge, Windsor, UK. IEEE Computer Society.

[40] Arcuri, A. and Yao, X. (2007d). Search Based Testing of Containers for Object-Oriented Software. Technical Report CSR-07-3, University of Birmingham.

[41] Arcuri, A. and Yao, X. (2008a). A Novel Co-evolutionary Approach to Automatic Software Bug Fixing. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '08)*, pages 162–168, Hongkong, China. IEEE Computer Society.

[42] Arcuri, A. and Yao, X. (2008b). Search Based Software Testing of Object-Oriented Containers. *Information Sciences*, **178**(15), 3075–3095.

[43] Arcuri, A., White, D. R., Clark, J., and Yao, X. (2008a). Multi-Objective Improvement of Software using Co-Evolution and Smart Seeding. In *Proceedings of the 7th International Conference on Simulated Evolution And Learning (SEAL '08)*, pages 61–70, Melbourne, Australia. Springer.

[44] Arcuri, A., Lehre, P. K., and Yao, X. (2008b). Theoretical Runtime Analyses of Search Algorithms on the Test Data Generation for the Triangle Classification Problem. In *Proceedings of the IEEE International Workshop on Search-Based Software Testing (SBST)*, pages 161–169, Lillehammer, Norway. IEEE Computer Society.

[45] Aversano, L., Di Penta, M., and Taneja, K. (2006). A Genetic Programming Approach to Support the Design Of Service Compositions. *Computer Systems Science & Engineering*, **21**(4), 247–254.

[46] Ayari, K., Bouktif, S., and Antoniol, G. (2007). Automatic Mutation Test Input Data Generation via Ant Colony. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1074–1081, London, England. ACM.

[47] Bagnall, A. J., Rayward-Smith, V. J., and Whittley, I. M. (2001). The Next Release Problem. *Information and Software Technology*, **43**(14), 883–890.

[48] Baker, P., Harman, M., Steinhöfel, K., and Skaliotis, A. (2006). Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *Proceedings of the 22nd IEEE International Conference*

on Software Maintenance (ICSM '06), pages 176–185, Philadelphia, Pennsylvania. IEEE Computer Society.

[49] Baradhi, G. and Mansour, N. (1997). A Comparative Study of Five Regression Testing Algorithms. In *Proceedings of the Australian Software Engineering Conference (ASWEC '97)*, pages 174–182, Sydney, NSW, Australia. IEEE Computer Society.

[50] Baresel, A. and Sthamer, H. (2003). Evolutionary Testing of Flag Conditions. In *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO '03)*, volume 2724 of *LNCS*, pages 2442–2454, Chicago, Illinois, USA. Springer.

[51] Baresel, A., Sthamer, H., and Schmidt, M. (2002). Fitness Function Design to Improve Evolutionary Structural Testing. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1329–1336, New York, USA. Morgan Kaufmann Publishers.

[52] Baresel, A., Pohlheim, H., and Sadeghipour, S. (2003). Structural and Functional Sequence Test of Dynamic and State-based Software with Evolutionary Algorithms. In *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO '03)*, volume 2724 of *LNCS*, pages 2428–2441, Chicago, Illinois, USA. Springer.

[53] Baresel, A., Sthamer, H., and Wegener, J. (2004a). Applying Evolutionary Testing to Search for Critical Defects. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1427–1428, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[54] Baresel, A., Binkley, D., Harman, M., and Korel, B. (2004b). Evolutionary Testing in the Presence of Loop-Assigned Flags: A Testability Transformation Approach. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '04)*, pages 108–118, Boston, Massachusetts, USA. ACM.

[55] Barlas, G. and El-Fakih, K. (2008). A GA-based Movie-On-Demand Platform using Multiple Distributed Servers. *Multimedia Tools and Applications*, **40**(3), 361–383.

[56] Barreto, A., de O. Barros, M., and Werner, C. M. (2008). Staffing a Software Project: a Constraint Satisfaction and Optimization-based Approach. *Computers & Operations Research*, **35**(10), 3073–3089.

[57] Bate, I. and Emberson, P. (2006). Incorporating Scenarios And Heuristics To Improve Flexibility In Real-Time Embedded Systems. In *Proceedings of the 12th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS '06)*, pages 221–230, San Jose, California, USA. IEEE.

[58] Baudry, B., Fleurey, F., Jézéquel, J.-M., and Traon, Y. L. (2002a). Automatic Test Cases Optimization using a Bacteriological Adaptation Model: Application to .NET Components. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE '02)*, pages 253–256, Edimburgh, UK. IEEE.

[59] Baudry, B., Fleurey, F., Jézéquel, J.-M., and Traon, Y. L. (2002b). Genes and Bacteria for Automatic Test Cases Optimization in the .NET Environment. In *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE '02)*, pages 195–206, Annapolis, MD, USA. IEEE Computer Society.

[60] Baudry, B., Fleurey, F., Jézéquel, J.-M., and Traon, Y. L. (2005a). Automatic Test Cases Optimization: a Bacteriologic Algorithm. *IEEE Software*, **22**(2), 76–82.

[61] Baudry, B., Fleurey, F., Jézéquel, J.-M., and Traon, Y. L. (2005b). From genetic to bacteriological algorithms for mutation-based testing: Research Articles. *Software Testing, Verification & Reliability*, **15**(2), 73–96.

[62] Berndt, D. J. and Watkins, A. (2004). Investigating the Performance of Genetic Algorithm-based Software Test Case Generation. In *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering*, pages 261–262, Tampa, Florida, USA. IEEE Computer Society.

[63] Berndt, D. J. and Watkins, A. (2005). High Volume Software Testing using Genetic Algorithms. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, page 318b, Big Island, Hawaii. IEEE Computer Society.

[64] Berndt, D. J., Fisher, J., Johnson, L., Pinglikar, J., and Watkins, A. (2003). Breeding Software Test Cases with Genetic Algorithms. In *Proceedings of the 36th Hawaii International Conference on System Sciences*, pages 338–348, Big Island, Hawaii. IEEE Computer Society.

[65] Beyer, H. G. and Sendhoff, B. (2007). Robustness optimization — a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, **196**, 3190–3218.

[66] Bhatia, R. K., Dave, M., and Joshi, R. C. (2008). Ant Colony based Rule Generation for Reusable Software Component Retrieval. In *Proceedings of the 1st Conference on India Software Engineering Conference (ISEC '08)*, pages 129–130, Hyderabad, India. ACM.

[67] Blanco, R., Tuya, J., Daz, E., and Daz, B. A. (2007). A Scatter Search Approach for Automated Branch Coverage in Software Testing. *International Journal of Engineering Intelligent Systems (EIS)*, **15**(3), 135–142.

[68] Bodhuin, T., Di Penta, M., and Troiano, L. (2007a). A Search-based Approach for Dynamically Re-packaging Downloadable Applications. In *Proceedings of the 2007 Conference of the IBM Center for Advanced Studies on Collaborative Research*, pages 27–41, Richmond Hill, Ontario, Canada. ACM.

[69] Bodhuin, T., Canfora, G., and Troiano, L. (2007b). SORMASA: A Tool for Suggesting Model Refactoring Actions by Metrics-Led Genetic Algorithm. In *Proceedings of the 1st Workshop on Refactoring Tools (WRT '07 - in conjunction with ECOOP'07)*, pages 23–24, Berlin, Germany. TU Berlin.

[70] Borgelt, K. (1998). *Software Test Data Generation from a Genetic Algorithm*, chapter Chapter 4 (Industrial Applications of Genetic Algorithms), pages 49–68. CRC Press.

[71] Boström, P. and Björkqvist, J. (2005). Optimisation-

based Black-Box Testing of Assertions in Simulink Models. Technical Report 711, bo Akademi University, Department of Computer Science, Turku Centre for Computer Science (TUCS), Turku, Finland.

[72] Bottaci, L. (2002). Instrumenting Programs with Flag Variables for Test Data Search by Genetic Algorithms. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1337–1342, New York, USA. Morgan Kaufmann Publishers.

[73] Bottaci, L. (2003). Predicate Expression Cost Functions to Guide Evolutionary Search for Test Data. In E. Cant-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO '03)*, volume 2724 of *LNCS*, pages 2455–2464, Chicago, Illinois, USA. Springer.

[74] Bouktif, S., Kégl, B., and Sahraoui, H. (2002). Combining Software Quality Predictive Models: An Evolutionary Approach. In *Proceedings of the International Conference on Software Maintenance (ICSM '02)*, pages 385–392, Montral, Canada. IEEE Computer Society.

[75] Bouktif, S., Azar, D., Precup, D., Sahraoui, H., and Kégl, B. (2004). Improving Rule Set Based Software Quality Prediction: A Genetic Algorithm-based Approach. *Journal of Object Technology*, **3**(4), 227–241.

[76] Bouktif, S., Antoniol, G., Merlo, E., and Neteler, M. (2006a). A Novel Approach to Optimize Clone Refactoring Activity. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1885–1892, Seattle, Washington, USA. ACM.

[77] Bouktif, S., Sahraoui, H., and Antoniol, G. (2006b). Simulated Annealing for Improving Software Quality Prediction. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1893–1900, Seattle, Washington, USA. ACM.

[78] Bowman, M., Briand, L. C., and Labiche, Y. (2008). Solving the Class Responsibility Assignment Problem in Object-Oriented Analysis with Multi-Objective Genetic Algorithms. Technical Report SCE-07-02, Carleton University.

[79] Briand, L. C., Feng, J., and Labiche, Y. (2002). Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE '02)*, pages 43–50, Ischia, Italy. ACM.

[80] Briand, L. C., Labiche, Y., and Shousha, M. (2005). Stress Testing Real-Time Systems with Genetic Algorithms. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1021–1028, Washington, D.C., USA. ACM.

[81] Briand, L. C., Labiche, Y., and Shousha, M. (2006). Using Genetic Algorithms for Early Schedulability Analysis and Stress Testing in Real-Time Systems. *Genetic Programming and Evolvable Machines*, **7**(2), 145–170.

[82] Bryce, R. C. and Colbourn, C. J. (2005). Constructing Interaction Test Suites with Greedy Algorithms. In D. F. Redmiles, T. Ellman, and A. Zisman, editors, *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05)*, pages 440–443, Long Beach, CA, USA. ACM.

[83] Bryce, R. C. and Colbourn, C. J. (2007a). One-Test-at-a-Time Heuristic Search for Interaction Test Suites. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1082–1089, London, England. ACM.

[84] Bryce, R. C. and Colbourn, C. J. (2007b). The Density Algorithm for Pairwise Interaction Testing. *Software Testing, Verification and Reliability*, **17**(3), 159–182.

[85] Bryce, R. C., Colbourn, C. J., and Cohen, M. B. (2005). A Framework of Greedy Methods for Constructing Interaction Test Suites. In G.-C. Roman, W. G. Griswold, and B. Nuseibeh, editors, *Proceedings of the 27th International Conference on Software Engineering (ICSE '05)*, pages 146–155, St. Louis, MO, USA. ACM.

[86] Bueno, P. M. S. and Jino, M. (2000). Identification of Potentially Infeasible Program Paths by Monitoring the Search for Test Data. In *Proceedings of the 15th IEEE international conference on Automated software engineering (ASE '00)*, pages 209–218, Grenoble, France. IEEE Computer Society.

[87] Bueno, P. M. S. and Jino, M. (2001). Automatic Test Data Generation for Program Paths Using Genetic Algorithms. In *Proceedings of the 13th International Conference on Software Engineering & Knowledge Engineering (SEKE '01)*, pages 2–9, Buenos Aires, Argentina.

[88] Bueno, P. M. S. and Jino, M. (2002). Automatic Test Data Generation for Program Paths using Genetic Algorithms. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, **12**(6), 691–709.

[89] Bueno, P. M. S., Wong, W. E., and Jino, M. (2007). Improving Random Test Sets using the Diversity Oriented Test Data Generation. In *Proceedings of the 2nd International Workshop on Random Testing*, pages 10–17, Atlanta, Georgia, USA. ACM.

[90] Bueno, P. M. S., Wong, W. E., and Jino, M. (2008). Automatic Test Data Generation using Particle Systems. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 809–814, Fortaleza, Ceara, Brazil. ACM.

[91] Bühler, O. and Wegener, J. (2008). Evolutionary Functional Testing. *Computers & Operations Research*, **35**(10), 3144–3160.

[92] Burgess, C. J. and Lefley, M. (2001). Can Genetic Programming Improve Software Effort Estimation? A Comparative Evaluation. *Information & Software Technology*, **43**(14), 863–873.

[93] Burke, E. and Kendall, G. (2005). *Search Methodologies. Introductory tutorials in optimization and decision support techniques*. Springer.

[94] Canfora, G., Di Penta, M., Esposito, R., and Villani,

M. L. (2004). A Lightweight Approach for QoS-Aware Service Composition. In *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC '04)*, New York, USA. ACM.

[95] Canfora, G., Di Penta, M., Esposito, R., and Villani, M. L. (2005a). An Approach for QoS-aware Service Composition based on Genetic Algorithms. In H.-G. Beyer and U.-M. O'Reilly, editors, *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1069–1075, Washington, D.C., USA. ACM.

[96] Canfora, G., Di Penta, M., Esposito, R., and Villani, M. L. (2005b). QoS-Aware Replanning of Composite Web Services. In *Proceedings of 2005 IEEE International Conference on Web Services (ICWS '05)*, pages 121–129, Orlando, FL, USA. IEEE Computer Society.

[97] Cao, L., Li, M., and Cao, J. (2005a). Cost-Driven Web Service Selection Using Genetic Algorithm. In *Proceedings of the 1st International Workshop on Internet and Network Economics (WINE '05)*, pages 906–915, Hong Kong, China. Springer.

[98] Cao, L., Cao, J., and Li, M. (2005b). Genetic Algorithm Utilized in Cost-Reduction Driven Web Service Selection. In *Proceedings of the International Conference on Computational Intelligence and Security (CIS '05)*, pages 679–686, Xi'an, China. Springer.

[99] Chan, B., Denzinger, J., Gates, D., Loose, K., and Buchanan, J. (2004). Evolutionary Behavior Testing of Commercial Computer Games. In *Proceedings of the 2004 Congress on Evolutionary Computation*, volume 1, pages 125–132, Portland, Oregon. IEEE.

[100] Chang, C. K. (1994). Changing Face of Software Engineering. *IEEE Software*, **11**(1), 4–5.

[101] Chang, C. K., Chao, C., Hsieh, S.-Y., and Alsalqan, Y. (1994). SPMNet: a Formal Methodology for Software Management. In *Proceedings of the 18th Annual International Computer Software and Applications Conference (COMPSAC '94)*, pages 57–57, Taipei, Taiwan. IEEE.

[102] Chang, C. K., Chao, C., Nguyen, T. T., and Christensen, M. (1998). Software Project Management Net: a new Methodology on Software Management. In *Proceedings of the 22nd Annual International Computer Software and Applications Conference (COMPSAC '98)*, pages 534–539, Vienna, Austria. IEEE Computer Science.

[103] Chang, C. K., Christensen, M. J., and Zhang, T. (2001). Genetic Algorithms for Project Management. *Annals of Software Engineering*, **11**(1), 107–139.

[104] Chang, C. K., yi Jiang, H., Di, Y., Zhu, D., and Ge, Y. (2008). Time-Line based Model for Software Project Scheduling with Genetic Algorithms. *Information and Software Technology*, **50**(11), 1142–1154.

[105] Chao, C., Komada, J., Liu, Q., Muteja, M., Alsalqan, Y., and Chang, C. (1993). An Application of Genetic Algorithms to Software Project Management. In *Proceedings of the 9th International Advanced Science and Technology*, pages 247–252, Chicago, Illinois, USA.

[106] Chardigny, S., Seriai, A., Tamzalit, D., and Oussalah, M. (2008a). Quality-Driven Extraction of a Component-based Architecture from an Object-Oriented System. In *Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR '08)*, pages 269–273, Athens, Greece. IEEE Computer Society.

[107] Chardigny, S., Seriai, A., Oussalah, M., and Tamzalit, D. (2008b). Search-based Extraction of Component-Based Architecture from Object-Oriented Systems. In *Proceedings of the 2nd European conference on Software Architecture*, volume 5292 of *LNCS*, pages 322–325, Paphos, Cyprus. Springer.

[108] Chen, Y. and Zhong, Y. (2008). Automatic Path-Oriented Test Data Generation Using a Multi-population Genetic Algorithm. In *Proceedings of the 4th International Conference on Natural Computation (ICNC '08)*, pages 565–570, Jinan, China. IEEE.

[109] Cheng, B. and Atlee, J. (2007). From state of the art to the future of requirements engineering. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*, Los Alamitos, California, USA. IEEE Computer Society Press. This volume.

[110] Cheon, Y. and Kim, M. (2006). A Specification-based Fitness Function for Evolutionary Testing of Object-oriented Programs. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1953–1954, Seattle, Washington, USA. ACM.

[111] Chicano, F. and Alba, E. (2008a). Ant Colony Optimization with Partial Order Reduction for Discovering Safety Property Violations in Concurrent Models. *Information Processing Letters*, **106**(6), 221–231.

[112] Chicano, F. and Alba, E. (2008b). Finding Liveness Errors with ACO. In *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI '08)*, pages 3002–3009, Hong Kong, China. IEEE Computer Society.

[113] Chicano, F. and Alba, E. (2008c). Searching for Liveness Property Violations in Concurrent Systems with ACO. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1727–1734, Atlanta, GA, USA. ACM.

[114] Clark, J. A. (2003). Nature-Inspired Cryptography: Past, Present and Future. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, volume 3, pages 1647–1654, Canberra, Australia. IEEE.

[115] Clark, J. A. and Jacob, J. L. (2000). Searching for a Solution: Engineering Tradeoffs and the Evolution of Provably Secure Protocols. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 82–95, Berkeley, California, USA. IEEE Computer Society.

[116] Clark, J. A. and Jacob, J. L. (2001). Protocols are Programs too: the Meta-Heuristic Search for Security Protocols. *Information & Software Technology*, **43**(14), 891–904.

[117] Clarke, J., Harman, M., Hierons, R. M., Jones, B., Lumkin, M., Rees, K., Roper, M., and Shepperd, M. J. (2000). The Application of Metaheuristic Search Techniques to Problems in Software Engineering. Technical Report TR-01-2000, University of York, Brunel University,

University of Glamorgan, British Telecom, Strathclyde University, Bournemouth University.

[118] Clarke, J., Dolado, J. J., Harman, M., Hierons, R. M., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., and Shepperd, M. J. (2003). Reformulating Software Engineering as A Search Problem. *IEE Proceedings - Software*, **150**(3), 161–175.

[119] Coffman, E. J., Garey, M., and Johnson, D. (1984). Approximation algorithms for bin-packing. *In Algorithm Design for Computer System Design*.

[120] Cohen, M., Kooi, S. B., and Srisa-an, W. (2006). Clustering the Heap in Multi-Threaded Applications for Improved Garbage Collection. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1901–1908, Seattle, Washington, USA. ACM.

[121] Cohen, M. B., Colbourn, C. J., and Ling, A. C. H. (2003a). Augmenting Simulated Annealing to Build Interaction Test Suites. In *Proceedings of the 14th International Symposium on Software Reliability Engineering*, pages 394–405, Denver, Colorado, USA. IEEE.

[122] Cohen, M. B., Gibbons, P. B., Mugridge, W. B., and Colbourn, C. J. (2003b). Constructing Test Suites for Interaction Testing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, pages 38–48, Portland, Oregon. IEEE Computer Society.

[123] Cohen, M. B., Dwyer, M. B., and Shi, J. (2007). Interaction Testing of Highly-Configurable Systems in the Presence of Constraints. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07)*, pages 129–139, London, United Kingdom. ACM.

[124] Cohen, M. B., Dwyer, M. B., and Shi, J. (2008). Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach. *IEEE Transactions on Software Engineering*, **34**(5), 633–650.

[125] Cooper, K. D., Schielke, P. J., and Subramanian, D. (1999). Optimizing for Reduced Code Space using Genetic Algorithms. In *Proceedings of the ACM Sigplan 1999 Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES '99)*, ACM Sigplan Notices, pages 1–9, Atlanta, Georgia, United States. ACM.

[126] Cornford, S. L., Dunphy, J., and Feather, M. S. (2002). Optimizing the Design of end-to-end Spacecraft Systems using Risk as a Currency. In *Proceedings of the IEEE Aerospace Conference*, pages 3361–3367, Big Sky, MT, USA. IEEE Computer Society.

[127] Cornford, S. L., Feather, M. S., Dunphy, J. R., Salcedo, J., and Menzies, T. (2003). Optimizing Spacecraft Design - Optimization Engine Development: Progress and Plans. In *Proceedings of the IEEE Aerospace Conference*, pages 3681–3690, Big Sky, Montana.

[128] Cortellessa, V., Marinelli, F., and Potena, P. (2008a). An Optimization Framework for "Build-Or-Buy" Decisions in Software Architecture. *Computers & Operations Research*, **35**(10), 3090–3106.

[129] Cortellessa, V., Crnkovic, I., Marinelli, F., and Potena, P. (2008b). Experimenting the Automated Selection of COTS Components Based on Cost and System Requirements. *Journal of Universal Computer Science*, **14**(8), 1228–1255.

[130] Cowan, G. S., Reynolds, R. G., and Cowan, G. J. (2004). *Acquisition of Software Engineering Knowledge SWEEP: An Automatic Programming System Based on Genetic Programming and Cultural Algorithms*, volume 14 of *Software Engineering and Knowledge Engineering*. World Scientific.

[131] Davies, E., McMaster, J., and Stark, M. (1994). The Use of Genetic Algorithms for Flight Test and Evaluation of Artificial Intelligence and Complex Software Systems. Technical Report AD-A284824, Naval Air Warfare Center, Patuxent River.

[132] de Abreu, B. T., Martins, E., and de Sousa, F. L. (2007). Generalized Extremal Optimization: An Attractive Alternative for Test Data Generation. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1138–1138, London, England. ACM.

[133] Del Grosso, C., Antoniol, G., Di Penta, M., Galinier, P., and Merlo, E. (2005). Improving Network Applications Security: A New Heuristic to Generate Stress Testing Data. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1037–1043, Washington, D.C., USA. ACM.

[134] Del Grosso, C., Antoniol, G., Merlo, E., and Galinier, P. (2008). Detecting Buffer Overflow via Automatic Test Input Data Generation. *Computers and Operations Research (COR) focused issue on Search Based Software Engineeering*, **35**(10), 3125–3143.

[135] Del Rosso, C. (2006). Reducing Internal Fragmentation in Segregated Free Lists using Genetic Algorithms. In *Proceedings of the 2006 International workshop on Workshop on Interdisciplinary Software Engineering Research (WISER '06)*, pages 57–60, Shanghai, China. ACM.

[136] Derderian, K., Hierons, R. M., Harman, M., and Guo, Q. (2005). Generating Feasible Input Sequences for Extended Finite State Machines (EFSMs) using Genetic Algorithms. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1081 – 1082, Washington, D.C., USA. ACM.

[137] Derderian, K., Hierons, R., Harman, M., and Guo, Q. (2006). Automated Unique Input Output Sequence Generation for Conformance Testing of FSMs. *The Computer Journal*, **49**(3), 331–344.

[138] Derderian, K. A. (2006). *Automated Test Sequence Generation for Finite State Machines using Genetic Algorithms*. Ph.D. thesis, School of Information Systems, Computing and Mathematics, Brunel University.

[139] Desnos, N., Huchard, M., Tremblay, G., Urtado, C., and Vauttier, S. (2008). Search-based Many-to-One Component Substitution. *Journal of Software Maintenance and Evolution: Research and Practice (Special Issue Search Based Software Engineering)*, **20**(5), 321–344.

[140] Di Penta, M. (2005). Evolution Doctor: A Framework to Control Software System Evolution. In *Proceedings of the 9th European Conference on Software Maintenance and*

Reengineering (CSMR '05), pages 280–283, Manchester, UK. IEEE Computer Society.

[141] Di Penta, M. and Taneja, K. (2005). Towards the Automatic Evolution of Reengineering Tools. In *Proceedings of the 9th European Conference on Software Maintenance and Reengineering (CSMR '05),*, pages 241–244, Manchester, UK. IEEE Computer Society.

[142] Di Penta, M., Neteler, M., Antoniol, G., and Merlo, E. (2005). A Language-Independent Software Renovation Framework. *Journal of Systems and Software*, **77**(3), 225–240.

[143] Di Penta, M., Canfora, G., Esposito, G., Mazza, V., and Bruno, M. (2007). Search-based Testing of Service Level Agreements. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1090–1097, London, England. ACM.

[144] Di Penta, M., Lombardi, P., Taneja, K., and Troiano, L. (2008). Search-based Inference of Dialect Grammars. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **12**(1), 51–66.

[145] Díaz, E., Tuya, J., and Blanco, R. (2003). Automated Software Testing using a Metaheuristic Technique based on Tabu Search. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE '03)*, pages 310–313, Montreal, Canada. IEEE.

[146] Díaz, E., Tuya, J., Blanco, R., and Dolado, J. J. (2008). A Tabu Search Algorithm for Structural Software Testing. *Computers & Operations Research*, **35**(10), 3052–3072.

[147] Dillon, E. (2005). *Hybrid Approach for the Automatic Determination of Worst Case Execution Time for Embedded Systems Written in C*. Master's thesis, Institute of Technology, Carlow.

[148] Dolado, J. J. (2000). A Validation of the Component-based Method for Software Size Estimation. *IEEE Transactions on Software Engineering*, **26**(10), 1006–1021.

[149] Dolado, J. J. (2001). On the Problem of the Software Cost Function. *Information and Software Technology*, **43**(1), 61–72.

[150] Dolado, J. J. and Fernandez, L. (1998). Genetic Programming, Neural Networks and Linear Regression in Software Project Estimation. In C. Hawkins, M. Ross, G. Staples, and J. B. Thompson, editors, *Proceedings of International Conference on Software Process Improvement, Research, Education and Training (INSPIRE III)*, pages 157–171, London, UK. British Computer Society.

[151] Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, **344**(2-3), 243–278.

[152] Doval, D., Mancoridis, S., and Mitchell, B. S. (1999). Automatic Clustering of Software Systems using a Genetic Algorithm. In *Proceedings of International Conference on Software Tools and Engineering Practice (STEP '99)*, pages 73–81, Pittsburgh, PA. IEEE.

[153] Dozier, G., Brown, D., Hurley, J., and Cain, K. (2004). Vulnerability Analysis of Immunity-based Intrusion Detection Systems using Evolutionary Hackers. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3102 of *Lecture Notes in Computer Science*, pages 263–274, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[154] Dozier, G., Brown, D., Hou, H., and Hurley, J. (2007). Vulnerability Analysis of Immunity-based Intrusion Detection Systems using Evolutionary Hackers. *Applied Soft Computing*, **7**(2), 547–553.

[155] El-Fakih, K., Yamaguchi, H., and v. Bochmann, G. (1999). A Method and a Genetic Algorithm for Deriving Protocols for Distributed Applications with Minimum Communication Cost. In *Proceedings of the 11th International Conference on Parallel and Distributed Computing and Systems (PDCS '99)*, Boston, USA.

[156] Emberson, P. and Bate, I. (2007). Minimising Task Migration and Priority Changes In Mode Transitions. In *Proceedings of the 13th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS '07)*, pages 158–167, Bellevue, Washington, USA. IEEE Computer Society.

[157] Emer, M. C. F. P. and Vergilio, S. R. (2002). GPTesT: A Testing Tool Based On Genetic Programming. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1343–1350, New York, USA. Morgan Kaufmann Publishers.

[158] Emer, M. C. F. P. and Vergilio, S. R. (2003). Selection and Evaluation of Test Data Based on Genetic Programming. *Software Quality Journal*, **11**(2), 167–186.

[159] Ernst, M. D. (2000). *Dynamically Discovering Likely Program Invariants*. PhD Thesis, University of Washington.

[160] Ernst, M. D., Cockrell, J., Griswold, W. G., and Notkin, D. (2001). Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, **27**(2), 1–25.

[161] Everson, R. M. and Fieldsend, J. E. (2006). Multiobjective Optimization of Safety Related Systems: An Application to Short-Term Conflict Alert. *IEEE Transactions on Evolutionary Computation*, **10**(2), 187–198.

[162] Evett, M. P., Khoshgoftaar, T. M., der Chien, P., and Allen, E. B. (1999). Using Genetic Programming to Determine Software Quality. In *Proceedings of the 12th International Florida Artificial Intelligence Research Society Conference (FLAIRS '99)*, pages 113–117, Orlando, FL, USA. Florida Research Society.

[163] Fatiregun, D., Harman, M., and Hierons, R. (2003). Search Based Transformations. In *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO '03)*, volume 2724 of *LNCS*, pages 2511–2512, Chicago, Illinois, USA. Springer.

[164] Fatiregun, D., Harman, M., and Hierons, R. M. (2004). Evolving Transformation Sequences using Genetic Algorithms. In *Proceedings of the Source Code Analysis and Manipulation, Fourth IEEE International Workshop*, pages 65–74, Chicago, Illinois, USA. IEEE Computer Society.

[165] Fatiregun, D., Harman, M., and Hierons, R. (2005). Search-based Amorphous Slicing. In *Proceedings of the*

12th International Working Conference on Reverse Engineering (WCRE '05), pages 3–12, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. IEEE Computer Society.

[166] Feather, M. S. and Menzies, T. (2002). Converging on the Optimal Attainment of Requirements. In *Proceedings of the 10th IEEE International Conference on Requirements Engineering (RE '02)*, pages 263–270, Essen, Germany. IEEE.

[167] Feather, M. S., Kiper, J. D., and Kalafat, S. (2004). Combining Heuristic Search, Visualization and Data Mining for Exploration of System Design Space. In *The International Council on Systems Engineering (INCOSE '04) - Proceedings of the 14th Annual International Symposium*, Toulouse, France.

[168] Feather, M. S., Cornford, S. L., Kiper, J. D., and Menzies, T. (2006). Experiences using Visualization Techniques to Present Requirements, Risks to Them, and Options for Risk Mitigation. In *Proceedings of the International Workshop on Requirements Engineering Visualization (REV '06)*, pages 10–10, Minnesota, USA. IEEE.

[169] Feldt, R. (1998a). An Experiment on using Genetic Programming to Develop Multiple Diverse Software Variants. Technical Report 98-13, Chalmers University of Technology, Gothenburg, Sweden.

[170] Feldt, R. (1998b). Forcing Software Diversity by Making Diverse Design Decisions - an Experimental Investigation. Technical Report 98-46, Chalmers University of Technology, Gothenburg, Sweden.

[171] Feldt, R. (1998c). Generating Multiple Diverse Software Versions with Genetic Programming. In *Proceedings of the 24th EUROMICRO Conference (EUROMICRO '98)*, volume 1, pages 387–394, Vesteras, Sweden. IEEE Computer Society.

[172] Feldt, R. (1998d). Generating Multiple Diverse Software Versions with Genetic Programming - an Experimental Study. *IEE Proceedings - Software Engineering*, **145**(6), 228–236.

[173] Feldt, R. (1998e). Using Genetic Programming to Systematically Force Software Diversity. Technical Report 296L, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

[174] Feldt, R. (1999). Genetic Programming as an Explorative Tool in Early Software Development Phases. In C. Ryan and J. Buckley, editors, *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering*, pages 11–20, University of Limerick, Ireland. Limerick University Press.

[175] Feldt, R. (2002). An Interactive Software Development Workbench based on Biomimetic Algorithms. Technical Report 02-16, Chalmers University of Technology, Gothenburg, Sweden.

[176] Feldt, R., Torkar, R., Gorschek, T., and Afzal, W. (2008). Searching for Cognitively Diverse Tests: Test Variability and Test Diversity Metrics. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 178–186, Lillehammer, Norway. IEEE Computer Society.

[177] Ferguson, R. and Korel, B. (1995). Software Test Data Generation Using the Chaining Approach. In *Proceedings of the IEEE International Test Conference on Driving Down the Cost of Test*, pages 703–709. IEEE Computer Society.

[178] Ferguson, R. and Korel, B. (1996). The Chaining Approach for Software Test Data Generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **5**(1), 63–86.

[179] Ferreira, L. P. and Vergilio, S. R. (2004). TDSGen: An Environment Based on Hybrid Genetic Algorithms for Generation of Test Data. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103/2004 of *Lecture Notes in Computer Science*, pages 1431–1432, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[180] Ferreira, M., Chicano, F., Alba, E., and Gómez-Pulido, J. A. (2008). Detecting Protocol Errors using Particle Swarm Optimization with Java Pathfinder. In W. W. Smari, editor, *Proceedings of the High Performance Computing & Simulation Conference (HPCS '08)*, pages 319–325, Nicosia, Cyprus.

[181] Finkelstein, A., Harman, M., Mansouri, S. A., Ren, J., and Zhang, Y. (2008). "Fairness Analysis" in Requirements Assignments. In *Proceedings of the 16th IEEE International Requirements Engineering Conference (RE '08)*, pages 115–124, Barcelona, Catalunya, Spain. IEEE Computer Society.

[182] Fischer, K. (1977). A Test Case Selection Method for the Validation of Software Maintenance Modifications. In *Proceedings of International Computer Software and Applications Conference (COMPSAC '77)*, pages 421–426, Chicago, USA.

[183] Fischer, K. F., Raji, F., and Chruscicki, A. (1981). A Methodology for Retesting Modified Software. In *Proceedings of the National Telecommunications Conference (NTC '81)*, pages 1–6, New Orleans, LA, USA.

[184] Funes, P., Bonabeau, E., Herve, J., and Morieux, Y. (2004). Interactive multi-participant task allocation. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1699–1705, Portland, Oregon. IEEE Press.

[185] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns*. Addison-Wesley.

[186] Garousi, V. (2006). *Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms*. Ph.D. thesis, Department of Systems and Computer Engineering, Carleton University.

[187] Garousi, V. (2008). Empirical Analysis of a Genetic Algorithm-based Stress Test Technique. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1743–1750, Atlanta, GA, USA. ACM.

[188] Garousi, V., Briand, L. C., and Labiche, Y. (2006). Traffic-aware Stress Testing of Distributed Real-Time Systems Based on UML Models using Genetic Algorithms. Technical Report TR SCE-06-09, Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S5B6, Canada.

[189] Garousi, V., Briand, L. C., and Labiche, Y. (2008). Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms. *Journal of Systems and Software*, **81**(2), 161–185.

[190] Ghani, K. and Clark, J. A. (2008). Strengthening Inferred Specification using Search Based Testing. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 187–194, Lillehammer, Norway. IEEE.

[191] Ghazi, S. and Ahmed, M. A. (2003). Pair-wise Test Coverage using Genetic Algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '03)*, pages 1420–1424, Canberra, Australia. IEEE.

[192] Ghiduk, A. S., Harrold, M. J., and Girgis, M. R. (2007). Using Genetic Algorithms to Aid Test-Data Generation for Data-Flow Coverage. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC '07)*, pages 41–48, Nagoya, Japan. IEEE.

[193] Girgis, M. R. (2005). Automatic Test Data Generation for Data Flow Testing using a Genetic Algorithm. *Journal of Universal Computer Science*, **11**(6), 898–915.

[194] Godefroid, P. (1997). Model Checking for Programming Languages using Verisoft. In *Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 174–186, Paris, France. ACM.

[195] Gold, N., Harman, M., Li, Z., and Mahdavi, K. (2006). Allowing Overlapping Boundaries in Source Code using a Search Based Approach to Concept Binding. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, pages 310–319, Philadelphia, USA. IEEE Computer Society.

[196] Goldsby, H. J. and Cheng, B. H. (2008a). Automatically Generating Behavioral Models of Adaptive Systems to Address Uncertainty. In *Proceedings of the 11th International conference on Model Driven Engineering Languages and Systems (MoDELS '08)*, pages 568–583, Toulouse, France. Springer.

[197] Goldsby, H. J. and Cheng, B. H. (2008b). Avida-MDE: a digital evolution approach to generating models of adaptive software behavior. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1751–1758, Atlanta, GA, USA. ACM.

[198] Goldsby, H. J., Cheng, B. H., McKinley, P. K., Knoester, D. B., and Ofria, C. A. (2008). Digital Evolution of Behavioral Models for Autonomic Systems. In *Proceedings of the 2008 International Conference on Autonomic Computing*, pages 87–96, Chicago, IL, USA. IEEE Computer Society.

[199] Gotlieb, A., Lazaar, N., and Lebbah, Y. (2008). Towards Constraint-based Local Search for Automatic Test Data Generation. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 195–195, Lillehammer, Norway. IEEE.

[200] Greer, D. and Ruhe, G. (2004). Software Release Planning: An Evolutionary and Iterative Approach. *Information & Software Technology*, **46**(4), 243–253.

[201] Groß, H.-G. (2000). *Measuring Evolutionary Testability of Real-Time Software*. Ph.D. thesis, University of Glamorgan.

[202] Groß, H.-G. (2001). A Prediction System for Evolutionary Testability applied to Dynamic Execution Time Analysis. *Information & Software Technology*, **43**(14), 855–862.

[203] Groß, H.-G. and Mayer, N. (2002). Evolutionary Testing in Component-based Real-Time System Construction. In E. Cantú-Paz, editor, *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 207–214, New York, USA. Morgan Kaufmann Publishers.

[204] Groß, H.-G. and Mayer, N. (2003). Search-based Execution-Time Verification in Object-Oriented and Component-Based Real-Time System Development. In *Proceedings of the 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '03)*, pages 113–120, Guadalajara, Mexico. IEEE Computer Society.

[205] Groß, H.-G., Jones, B. F., and Eyres, D. E. (2000). Structural Performance Measure of Evolutionary Testing applied to Worst-Case Timing of Real-Time Systems. *IEE Proceedings - Software*, **147**(2), 25–30.

[206] Guo, Q. (2006). *Improving Fault Coverage and Minimising the Cost of Fault Identification When Testing from Finite State Machines*. Ph.D. thesis, School of Information Systems, Computing and Mathematics, Brunel University.

[207] Guo, Q., Harman, M., Hierons, R., and Derderian, K. (2003). Computing Unique Input/Output Sequences using Genetic Algorithms. In *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES '03)*, volume 2931, pages 164–177, Montreal, Quebec, Canada. Springer.

[208] Guo, Q., Hierons, R. M., Harman, M., and Derderian, K. (2005). Constructing Multiple Unique Input/Output Sequences using Evolutionary Optimisation Techniques. *IEE Proceedings - Software*, **152**(3), 127–140.

[209] Guo, Q., Hierons, R. M., Harman, M., and Derderian, K. (2007). Heuristics for Fault Diagnosis when Testing from Finite State Machines. *Software Testing, Verification & Reliability*, **17**(1), 41–57.

[210] Gupta, N. K. and Rohil, M. K. (2008). Using Genetic Algorithm for Unit Testing of Object Oriented Software. In *Proceedings of the 1st International Conference on Emerging Trends in Engineering and Technology (ICETET '08)*, pages 308–313, Nagpur, India. IEEE.

[211] Haas, J., Peysakhov, M., and Mancoridis, S. (2005). GA-based Parameter Tuning for Multi-Agent Systems. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1085–1086, Washington, D.C., USA. ACM.

[212] Hanh, V. L., Akif, K., Traon, Y. L., and Jézéque, J.-M. (2001). Selecting an Efficient OO Integration Testing Strategy: An Experimental Comparison of Actual Strategies. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP 2001)*, volume 2072/2001, pages 381–401, Budapest, Hungary. Springer.

[213] Harman, M. (2006). Search-based Software Engineer-

ing for Maintenance and Reengineering. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR '06)*, page 311, Bari, Italy. IEEE Computer Society.

[214] Harman, M. (2007a). Search Based Software Engineering for Program Comprehension. In *Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 3–13, Banff, Alberta, Canada. IEEE.

[215] Harman, M. (2007b). The Current State and Future of Search Based Software Engineering. In L. Briand and A. Wolf, editors, *Proceedings of International Conference on Software Engineering / Future of Software Engineering 2007 (ICSE/FOSE '07)*, pages 342–357, Minneapolis, Minnesota, USA. IEEE Computer Society.

[216] Harman, M. (2008). Testability Transformation for Search-Based Testing. In *Keynote of the 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, Lillehammer, Norway.

[217] Harman, M. and Clark, J. A. (2004). Metrics Are Fitness Functions Too. In *Proceedings of the 10th IEEE International Symposium on Software Metrics (METRICS '04)*, pages 58–69, Chicago, USA. IEEE Computer Society.

[218] Harman, M. and Jones, B. F. (2001a). Search-based Software Engineering. *Information & Software Technology*, **43**(14), 833–839.

[219] Harman, M. and Jones, B. F. (2001b). Software Engineering using Metaheuristic Innovative Algorithms: Workshop Report. *Information and Software Technology*, **43**(14), 762–763.

[220] Harman, M. and Jones, B. F. (2001c). The SEMINAL Workshop: Reformulating Software Engineering as a Metaheuristic Search Problem. *ACM SIGSOFT Software Engineering Notes*, **26**(6), 62–66.

[221] Harman, M. and McMinn, P. (2007). A Theoretical & Empirical Analysis of Evolutionary Testing and Hill Climbing for Structural Test Data Generation. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2007)*, pages 73–83, London, England. ACM.

[222] Harman, M. and Tratt, L. (2007). Pareto Optimal Search Based Refactoring at the Design Level. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1106–1113, London, England. ACM.

[223] Harman, M. and Wegener, J. (2004). Getting Results from Search-Based Approaches to Software Engineering. In *Proceedings of the 26th International Conference on Software Engineering (ICSE '04)*, pages 728–729, Edinburgh, Scotland, UK. IEEE Computer Society.

[224] Harman, M., Hierons, R., and Proctor, M. (2002a). A New Representation and Crossover Operator for Search-based Optimization of Software Modularization. In *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1351–1358, New York, USA. Morgan Kaufmann Publishers.

[225] Harman, M., Hu, L., Hierons, R. M., Fox, C., Danicic, S., Baresel, A., Sthamer, H., and Wegener, J. (2002b). Evolutionary Testing Supported by Slicing and Transformation. In *Proceedings of IEEE International Conference on Software Maintenance (ICSM '02)*, pages 285–285, Montreal, Canada. IEEE.

[226] Harman, M., Hu, L., Hierons, R., Baresel, A., and Sthamer, H. (2002c). Improving Evolutionary Testing by Flag Removal. In *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1359–1366, New York, USA. Morgan Kaufmann Publishers.

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

[228] Harman, M., Swift, S., and Mahdavi, K. (2005). An Empirical Study of the Robustness of Two Module Clustering Fitness Functions. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, volume 1, pages 1029–1036, Washington, D.C., USA. ACM.

[229] Harman, M., Skaliotis, A., and Steinhfel, K. (2006). Search-based Approaches to the Component Selection and Prioritization Problem. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1951–1952, Seattle, Washington, USA. ACM.

[230] Harman, M., Lakhotia, K., and McMinn, P. (2007a). A Multi-Objective Approach to Search-based Test Data Generation. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1098–1105, London, England. ACM.

[231] Harman, M., Hassoun, Y., Lakhotia, K., McMinn, P., and Wegener, J. (2007b). The Impact of Input Domain Reduction on Search-based Test Data Generation. In *Proceedings of the the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, pages 155–164, Dubrovnik, Croatia. ACM.

[232] Harman, M., Islam, F., Xie, T., and Wappler, S. (2009). Automated Test Data Generation for Aspect-Oriented Programs. In *Proceedings of the 8th International Conference on Aspect-Oriented Software Development (AOSD '09)*, pages 185–196, Charlottesville, Virginia, USA. ACM.

[233] Hart, J. and Shepperd, M. J. (2002). Evolving Software with Multiple Outputs and Multiple Populations. In *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 223–227, New York, USA. Morgan Kaufmann Publishers.

[234] Hartmann, J. and Robson, D. J. (1989). Revalidation during the Software Maintenance Phase. In *Proceedings of the 1989 Conference on Software Maintenance*, pages 70–80, Miami, Florida, USA. IEEE.

[235] Hartmann, J. and Robson, D. J. (1990a). Retest-Development of A Selective Revalidation Prototype Environment for Use In Software Maintenance. In *Proceedings of the 23rd Annual Hawaii International Conference on System Sciences*, volume 2, pages 92–101, Kailua-Kona, HI, USA. IEEE.

[236] Hartmann, J. and Robson, D. J. (1990b). Techniques

for Selective Revalidation. *IEEE Software*, **7**(1), 31–36.

[237] He, P., Kang, L., and Fu, M. (2008). Formality Based Genetic Programming. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '08) (IEEE World Congress on Computational Intelligence)*, pages 4080–4087, Hong Kong, China. IEEE Press.

[238] Hericko, M., Zivkovic, A., and Rozman, I. (2008). An Approach to Optimizing Software Development Team Size. *Information Processing Letters*, **108**(3), 101–106.

[239] Hermadi, I. (2004). *Genetic Algorithm based Test Data Generator*. Master's thesis, King Fahd University of Petroleum & Minerals, Information & Computer Science Department, Dhahran, Saudi Arabia.

[240] Hermadi, I. and Ahmed, M. A. (2003). Genetic Algorithm based Test Data Generator. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC '03)*, pages 85–91, Canberra, Australia. IEEE Press.

[241] Hierons, R. M., Harman, M., Guo, Q., and Dederian, K. (2004). Input Sequence Generation for Testing of Communicating Finite State Machines (CFSMs). In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103/2004 of *Lecture Notes in Computer Science*, pages 1429–1430, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[242] Hierons, R. M., Harman, M., and Fox, C. (2005). Branch-Coverage Testability Transformation for Unstructured Programs. *Computer Journal*, **48**(4), 421–436.

[243] Hla, K. H. S., Choi, Y., and Park, J. S. (2008). Applying Particle Swarm Optimization to Prioritizing Test Cases for Embedded Real Time Software. In *Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, pages 527–532, Sydney, Australia. IEEE Computer Society.

[244] Hodjat, B., Ito, J., and Amamiya, M. (2004). A Genetic Algorithm to Improve Agent-Oriented Natural Language Interpreters. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1307–1309, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[245] Holland, J. H. (1975). *Adaption in Natural and Artificial Systems*. MIT Press, Ann Arbor.

[246] Hoste, K. and Eeckhout, L. (2008). COLE: Compiler Optimization Level Exploration. In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, pages 165–174, Boston, MA, USA. ACM.

[247] Hsu, C.-J., Huang, C.-Y., and Chen, T.-Y. (2008). A Modified Genetic Algorithm for Parameter Estimation of Software Reliability Growth Models. In *Proceedings of the 19th International Symposium on Software Reliability Engineering (ISSRE '08)*, pages 281–282, Seattle/Redmond, WA, USA. IEEE.

[248] Huang, S.-J., Chiu, N.-H., and Chen, L.-W. (2008). Integration of the Grey Relational Analysis with Genetic Algorithm for Software Effort Estimation. *European Journal of Operational Research*, **188**(3), 898–909.

[249] Huynh, S. and Cai, Y. (2007). An Evolutionary Approach to Software Modularity Analysis. In *Proceedings of the 1st International Workshop on Assessment of Contemporary Modularization Techniques (ACoM'07)*, pages 1–6, Minneapolis, USA. ACM.

[250] Jaeger, M. C. and Mühl, G. (2007). QoS-based Selection of Services: The Implementation of a Genetic Algorithm. In *Proceedings of Kommunikation in Verteilten Systemen (KiVS) 2007 Workshop: Service-Oriented Architectures and Service-Oriented Computing*.

[251] Jalali, O., Menzies, T., and Feather, M. (2008). Optimizing Requirements Decisions With KEYS. In *PROMISE'08*, pages 79–86, Leipzig, Germany. ACM.

[252] Jarillo, G., Succi, G., Pedrycz, W., and Reformat, M. (2001). Analysis of Software Engineering Data using Computational Intelligence Techniques. In *Proceedings of the 7th International Conference on Object Oriented Information Systems (OOIS '01)*, pages 133–142, Calgary, Canada. Springer.

[253] Jia, Y. and Harman, M. (2008a). Constructing Subtle Faults Using Higher Order Mutation Testing. In *Proceedings of the 8th International Working Conference on Source Code Analysis and Manipulation (SCAM '08)*, pages 249–258, Beijing, China. IEEE.

[254] Jia, Y. and Harman, M. (2008b). MILU : A Customizable, Runtime-Optimized Higher Order Mutation Testing Tool for the Full C Language. In *Proceedings of the 3rd Testing: Academic and Industrial Conference - Practice and Research Techniques (TAIC PART '08)*, pages 94–98, Windsor, UK. IEEE.

[255] Jiang, H. (2006). Can the Genetic Algorithm Be a Good Tool for Software Engineering Searching Problems? In *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC '06)*, pages 362–366, Chicago, USA. IEEE Computer Society.

[256] Jiang, H., Chang, C. K., Zhu, D., and Cheng, S. (2007a). A Foundational Study on the Applicability of Genetic Algorithm to Software Engineering Problems. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC '07)*, pages 2210–2219, Singapore. IEEE.

[257] Jiang, T., Gold, N., Harman, M., and Li, Z. (2007b). Locating Dependence Structures using Search-based Slicing. *Information and Software Technology*, **50**(12), 1189–1209.

[258] Jiang, T., Harman, M., and Hassoun, Y. (2008). Analysis of Procedure Splitability. In *Proceedings of the 15th Working Conference on Reverse Engineering (WCRE '08)*, pages 247–256, Antwerp, Belgium. IEEE Computer Society.

[259] Johnson, C. (2007). Genetic Programming with Fitness based on Model Checking. In *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *LNCS*, pages 114–124, Valencia, Spain. Springer.

[260] Jones, B. F., Sthamer, H.-H., and Eyres, D. E. (1995a). Generating Test Data for Ada Procedures using Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA '95)*, pages 65–70, London, UK. IEE.

[261] Jones, B. F., Sthamer, H.-H., Yang, X., and Eyres, D. E.

(1995b). The Automatic Generation of Software Test Data Sets using Adaptive Search Techniques. *Transactions on Information and Communications Technologies: Software Quality Management*, **11**, 435–444.

[262] Jones, B. F., Sthamer, H.-H., and Eyres, D. E. (1996). Automatic Structural Testing using Genetic Algorithms. *Software Engineering Journal*, **11**(5), 299–306.

[263] Jones, B. F., Eyres, D. E., and Sthamer, H.-H. (1998). A Strategy for using Genetic Algorithms to Automate Branch and Fault-based Testing. *Computer Journal*, **41**(2), 98–107.

[264] Joshi, A. M., Eeckhout, L., John, L. K., and Isen, C. (2008). Automated Microprocessor Stressmark Generation. In *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture (HPCA '08)*, pages 229–239, Salt Lake City, UT, USA. IEEE.

[265] Kalaji, A., Hierons, R. M., and Swift, S. (2008). Automatic Generation of Test Sequences form EFSM Models using Evolutionary Algorithms. Technical report, School of Information Systems, Computing and Mathematics, Brunel University.

[266] Kapur, P., Ngo-The, A., Ruhe, G., and Smith, A. (2008). Optimized staffing for product releases and its application at Chartwell Technology. *Journal of Software Maintenance and Evolution: Research and Practice (Special Issue Search Based Software Engineering)*, **20**(5), 365–386.

[267] Karlsson, J., Wohlin, C., and Regnell, B. (1998). An evaluation of methods for priorizing software requirements. *Information and Software Technology*, **39**, 939–947.

[268] Katz, G. and Peled, D. (2008a). Genetic Programming and Model Checking: Synthesizing New Mutual Exclusion Algorithms. In *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA '08)*, volume 5311 of *LNCS*, pages 33–47, Seoul, Korea. Springer.

[269] Katz, G. and Peled, D. (2008b). Model Checking-Based Genetic Programming with an Application to Mutual Exclusion. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08)*, pages 141–156, Budapest, Hungary. Springer.

[270] Kessentini, M., Sahraoui, H., and Boukadoum, M. (2008). Model Transformation as an Optimization Problem. In *Proceedings of the ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (MODELS '08)*, volume 5301 of *Lecture Notes in Computer Science*, pages 159–173, Toulouse, France. Springer.

[271] Khamis, A. M., Girgis, M. R., and Ghiduk, A. S. (2007). Automatic Software Test Data Generation for Spanning Sets Coverage using Genetic Algorithms. *Computing and Informatics*, **26**(4), 383–401.

[272] Khoshgoftaar, T. M. and Liu, Y. (2007). A Multi-Objective Software Quality Classification Model Using Genetic Programming. *IEEE Transactions On Reliability*, **56**(2), 237–245.

[273] Khoshgoftaar, T. M., Liu, Y., and Seliya, N. (2003). Genetic Programming-based Decision Trees for Software Quality Classification. In *Proceedings of the 15th International Conference on Tools with Artificial Intelligence (ICTAI '03)*, pages 374–383, Sacramento, California, USA. IEEE Computer Society.

[274] Khoshgoftaar, T. M., Liu, Y., and Seliya, N. (2004a). A Multiobjective Module-Order Model for Software Quality Enhancement. *IEEE Transactions on Evolutionary Computation*, **8**(6), 593–608.

[275] Khoshgoftaar, T. M., Liu, Y., and Seliya, N. (2004b). Module-Order Modeling using an Evolutionary Multi-Objective Optimization Approach. In *Proceedings of the 10th IEEE International Symposium on Software Metrics (METRICS '04)*, pages 159–169, Chicago, USA. IEEE Computer Society.

[276] Khoshgoftaar, T. M., Seliya, N., and Drown, D. J. (2008). On the Rarity of Fault-prone Modules in Knowledge-based Software Quality Modeling. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering*, pages 279–284, San Francisco, CA, USA. Knowledge Systems Institute Graduate School.

[277] Khurshid, S. (2001). Testing an Intentional Naming Scheme using Genetic Algorithms. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '01)*, volume 2031 of *Lecture Notes In Computer Science*, pages 358–372, Genova, Italy. Springer.

[278] King, K. N. and Offutt, A. J. (1991). A FORTRAN language system for mutation-based software testing. *Software Practice and Experience*, **21**, 686–718.

[279] Kiper, J. D., Feather, M. S., and Richardson, J. (2007). Optimizing the V&V Process for Critical Systems. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1139–1139, London, England. ACM.

[280] Kirsopp, C., Shepperd, M., and Hart, J. (2002). Search Heuristics, Case-based Reasoning And Software Project Effort Prediction. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1367–1374, New York. Morgan Kaufmann Publishers.

[281] Korel, B. (1990). Automated Software Test Data Generation. *Transactions on Software Engineering*, **SE-16**(8), 870–879.

[282] Korel, B., Chung, S., and Apirukvorapinit, P. (2003). Data Dependence Analysis in Automated Test Generation. In M. Hamza, editor, *Proceedings of International Conference on Software Engineering and Applications (SEA 2003)*, pages 476–481, Marina del Rey, USA. ACTA.

[283] Korel, B., Harman, M., Chung, S., Apirukvorapinit, P., Gupta, R., and Zhang, Q. (2005). Data Dependence Based Testability Transformation in Automated Test Generation. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE '05)*, pages 245–254, Chicago, Illinios, USA. IEEE Computer Society.

[284] Koza, J. R. (1992). *Genetic Programming: On the*

Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA.

[285] Kuperberg, M., Krogmann, K., and Reussner, R. (2008). Performance Prediction for Black-Box Components Using Reengineered Parametric Behaviour Models. In *Proceedings of the 11th International Symposium on Component-Based Software Engineering (CBSE '08)*, volume 5282 of *LNCS*, pages 48–63, Karlsruhe, Germany. Springer.

[286] Lakhotia, K., Harman, M., and McMinn, P. (2008). Handling Dynamic Data Structures in Search Based Testing. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1759–1766, Atlanta, GA, USA. ACM.

[287] Lam, C. P., Xiao, J., and Li, H. (2007). Ant Colony Optimisation for Generation of Conformance Testing Sequences using a Characterising Set. In *Proceedings of the 3rd IASTED Conference on Advances in Computer Science and Technology*, pages 140–146, Phuket, Thailand. ACTA Press.

[288] Lammermann, F. and Wappler, S. (2005). Benefits of Software Measures for Evolutionary White-Box Testing. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1083–1084, Washington, D.C., USA. ACM.

[289] Lammermann, F., Baresel, A., and Wegener, J. (2004). Evaluating Evolutionary Testability with Software-Measurements. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1350–1362, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[290] Lange, R. and Mancoridis, S. (2007). Using Code Metric Histograms and Genetic Algorithms to Perform Author Identification for Software Forensics. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 2082–2089, London, England. ACM.

[291] Lefley, M. and Shepperd, M. J. (2003). Using Genetic Programming to Improve Software Effort Estimation Based on General Data Sets. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO '03)*, volume 2724 of *LNCS*, pages 2477–2487, Chicago, Illinois, USA. Springer.

[292] Lefticaru, R. and Ipate, F. (2007). Automatic State-based Test Generation using Genetic Algorithms. In *Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '07)*, pages 188–195, Timisoara, Romania. IEEE Computer Society.

[293] Lefticaru, R. and Ipate, F. (2008a). Functional Search-based Testing from State Machines. In *Proceedings of the First International Conference on Software Testing, Verfication and Validation (ICST 2008)*, pages 525–528, Lillehammer, Norway. IEEE Computer Society.

[294] Lefticaru, R. and Ipate, F. (2008b). Search-based Testing using State-based Fitness. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 210–210, Lillehammer, Norway. IEEE.

[295] Lehre, P. K. and Yao, X. (2007). Runtime Analysis of (1+1) EA on Computing Unique Input Output Sequences. In *Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC '07)*, pages 1882–1889, Singapore. IEEE.

[296] Lehre, P. K. and Yao, X. (2008). Crossover can be Constructive when Computing Unique Input Output Sequences. In *Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL '08)*, volume 5361 of *LNCS*, pages 595–604, Melbourne, Australia. Springer.

[297] Levin, S. and Yehudai, A. (2007). Evolutionary Testing: A Case Study. *Hardware and Software, Verification and Testing*, **4383/2007**, 155–165.

[298] Li, H. and Lam, C. P. (2003). An Evolutionary Approach for Optimisation of State-based Test Suites for Software Systems. In W. Dosch and R. Y. Lee, editors, *Proceedings of the 4th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD '03)*, pages 226–233, Lübeck, Germany. ACIS.

[299] Li, H. and Lam, C. P. (2005a). An Ant Colony Optimization Approach to Test Sequence Generation for Statebased Software Testing. In *Proceedings of the 5th International Conference on Quality Software (QSIC '05)*, pages 255–264, Melbourne, Australia. IEEE Computer Society.

[300] Li, H. and Lam, C. P. (2005b). Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams. In F. Khendek and R. Dssouli, editors, *Proceedings of Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference (TestCom '05)*, volume 3502 of *Lecture Notes in Computer Science*, pages 69–80, Montreal, Canada. Springer.

[301] Li, K. and Wu, M. (2004). *Effective Software Test Automation: Developing an Automated Software Testing Tool*. Sybex.

[302] Li, Z., Harman, M., and Hierons, R. M. (2007). Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering*, **33**(4), 225–237.

[303] Liaskos, K. and Roper, M. (2007). Automatic Test-Data Generation: An Immunological Approach. In *Proceedings of Testing: Accademic and Industrial Conference - Practice and Research Techniques (TAIC PART '07)*, pages 77–81, Windsor, UK. IEEE.

[304] Liaskos, K. and Roper, M. (2008). Hybridizing Evolutionary Testing with Artificial Immune Systems and Local Search. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 211–220, Lillehammer, Norway. IEEE.

[305] Liaskos, K., Roper, M., and Wood, M. (2007). Investigating Data-Flow Coverage of Classes Using Evolutionary Algorithms. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*,

pages 1140–1140, London, England. ACM.

[306] Lin, J.-C. and Yeh, P.-L. (2000). Using Genetic Algorithms for Test Case Generation in Path Testing. In *Proceedings of the 9th Asian Test Symposium (ATS '00)*, pages 241–246, Taipei, Taiwan. IEEE.

[307] Lin, J.-C. and Yeh, P.-L. (2001). Automatic Test Data Generation for Path Testing using GAs. *Information Sciences*, **131**(1-4), 47–64.

[308] Linden, D. S. (2002). Innovative antenna design using genetic algorithms. In D. W. Corne and P. J. Bentley, editors, *Creative Evolutionary Systems*, chapter 20. Elsevier, Amsterdam, The Netherland.

[309] Liu, X., Liu, H., Wang, B., Chen, P., and Cai, X. (2005a). A Unified Fitness Function Calculation Rule for Flag Conditions to Improve Evolutionary Testing. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05)*, pages 337–341, Long Beach, CA, USA. ACM.

[310] Liu, X., Wang, B., and Liu, H. (2005b). Evolutionary Search in the context of Object-Oriented Programs. In *Proceedings of the 6th Metaheuristics International Conference (MIC '05)*, Vienna, Austria.

[311] Liu, X., Wang, L., Zhu, X., Bai, Z., Zhang, M., and Liu, H. (2007a). Fitness Calculation Approach for Nested If-else Construct in Evolutionary Testing. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1141–1141, London, England. ACM.

[312] Liu, X., Zhang, M., Bai, Z., Wang, L., Du, W., and Wang, Y. (2007b). Function Call Flow based Fitness Function Design in Evolutionary Testing. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC '07)*, pages 57–64, Nagoya, Japan. IEEE Computer Society.

[313] Liu, Y. and Khoshgoftaar, T. (2004). Reducing Overfitting in Genetic Programming Models for Software Quality Classification. In *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE '04)*, pages 56–65, Tampa, Florida, USA. IEEE Computer Society.

[314] Liu, Y. and Khoshgoftaar, T. M. (2001). Genetic Programming Model for Software Quality Classification. In *Proceedings of the 6th IEEE International Symposium on High-Assurance Systems Engineering: Special Topic: Impact of Networking (HASE '01)*, pages 127–136, Boco Raton, FL, USA. IEEE Computer Society.

[315] Liu, Y. and Khoshgoftaar, T. M. (2003). Building Decision Tree Software Quality Classification Models Using Genetic Programming. In *Proccedings of the Genetic and Evolutionary Computation Conference (GECCO '03)*, volume 2724 of *LNCS*, pages 1808–1809, Chicago, Illinois, USA. Springer.

[316] Lokan, C. (2005). What Should You Optimize When Building an Estimation Model? In *Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS '05)*, pages 34–44, Como, Italy. IEEE Computer Society.

[317] Lucas, S. M. and Reynolds, T. J. (2005). Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**(7), 1063–1074.

[318] Lutz, R. (2001). Evolving Good Hierarchical Decompositions of Complex Systems. *Journal of Systems Architecture*, **47**(7), 613–634.

[319] Ma, Y. and Zhang, C. (2008). Quick Convergence of Genetic Algorithm for QoS-driven Web Service Selection. *Computer Networks*, **52**(5), 1093–1104.

[320] MacNish, C. (2000). Evolutionary Programming Techniques for Testing Students' Code. In *Proceedings of the 4th Australasian Conference on Computer Science Education (ACSE '00)*, pages 170–173, Melbourne, Australia. ACM.

[321] Mahanti, P. K. and Banerjee, S. (2006). Automated Testing in Software Engineering: using Ant Colony and Self-Regulated Swarms. In *Proceedings of the 17th IASTED international conference on Modelling and simulation (MS '06)*, pages 443–448, Montreal, Canada. ACTA Press.

[322] Mahdavi, K. (2005). *A Clustering Genetic Algorithm for Software Modularisation with Multiple Hill Climbing Approach*. Ph.D. thesis, Brunel University West London.

[323] Mahdavi, K., Harman, M., and Hierons, R. M. (2003a). A Multiple Hill Climbing Approach to Software Module Clustering. In *Proceedings of the International Conference on Software Maintenance (ICSM '03)*, pages 315–324, Amsterdam, Holland. IEEE Computer Society.

[324] Mahdavi, K., Harman, M., and Hierons, R. (2003b). Finding Building Blocks for Software Clustering. In *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO '03)*, volume 2724 of *LNCS*, pages 2513–2514, Chicago, Illinois, USA. Springer.

[325] Makai, M. C. (2008). *Incorporating Design Knowledge into Genetic Algorithm-based White-Box Software Test Case Generators*. Master's thesis, Virginia Tech.

[326] Mancoridis, S., Mitchell, B. S., Rorres, C., Chen, Y., and Gansner, E. R. (1998). Using Automatic Clustering to Produce High-Level System Organizations of Source Code. In *Proceedings of the 6th International Workshop on Program Comprehension (IWPC '98)*, pages 45–52, Ischia, Italy. IEEE Computer Society Press.

[327] Mancoridis, S., Mitchell, B. S., Chen, Y., and Gansner, E. R. (1999). Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99)*, pages 50–59, Oxford, England, UK. IEEE.

[328] Mansour, N. and Bahsoon, R. (2002). Reduction-based Methods and Metrics for Selective Regression Testing. *Information and Software Technology*, **44**(7), 431–443.

[329] Mansour, N. and El-Fakih, K. (1999). Simulated Annealing and Genetic Algorithms for Optimal Regression Testing. *Journal of Software Maintenance: Research and Practice*, **11**(1), 19–34.

[330] Mansour, N. and Salame, M. (2004). Data Generation for Path Testing. *Software Quality Control*, **12**(2), 121–136.

[331] Mansour, N., Bahsoon, R., and Baradhi, G. (2001). Empirical Comparison of Regression Test Selection Algorithms. *Journal of Systems and Software*, **57**(1), 79–90.

[332] Mantere, T. (2003). *Automatic Software Testing by Genetic Algorithms*. Ph.D. thesis, University of Vaasa.

[333] Mantere, T. and Alander, J. T. (2001). Automatic Software Testing by Genetic Algorithms - Introduction to Method and Consideration of Possible Pitfalls. In *Proceedings of the 7th International Mendel Conferennce on Soft Computing (MENDEL '01)*, pages 19–23, Brno, Czech Republic. Kuncik, Brno, Check Republic.

[334] Mantere, T. and Alander, J. T. (2005). Evolutionary Software Engineering, A Review. *Applied Soft Computing*, **5**(3), 315–331.

[335] Masud, M. M., Nayak, A., Zaman, M., and Bansal, N. (2005). Strategy for Mutation Testing using Genetic Algorithms. In *Proceedings of 2005 Canadian Conference on Electrical and Computer Engineering*, pages 1049–1052, Saskatoon, Saskatchewan Canada. IEEE.

[336] May, P., Mander, K., and Timmis, J. (2003). Software Vaccination: An Artificial Immune System Approach to Mutation Testing. In *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS '03)*, volume 2787 of *Lecture Notes in Computer Science*, pages 81–92, Edinburgh, UK. Springer.

[337] May, P., Timmis, J., and Mander, K. (2007). Immune and Evolutionary Approaches to Software Mutation Testing. In *Proceedings of the 6th International Conference on Artificial Immune Systems (ICARIS '07)*, pages 336–347, Santos, Brazil. Springer.

[338] Mayer, J. (2006). Towards Effective Adaptive Random Testing for Higher-Dimensional Input Domains. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1955–1956, Seattle, Washington, USA. ACM.

[339] McGraw, G., Michael, C., and Schatz, M. (1998). Generating Software Test Data by Evolution. Technical Report RSTR-018-97-01, Reliable Software Technologies, Sterling, VA.

[340] McMinn, P. (2004). Search-based Software Test Data Generation: A Survey. *Software Testing, Verification and Reliability*, **14**(2), 105–156.

[341] McMinn, P. and Holcombe, M. (2003). The State Problem for Evolutionary Testing. In *Proceedings of the 2003 Conference on Genetic and Evolutionary Computation (GECCO '03)*, volume 2724 of *LNCS*, pages 2488–2498, Chicago, Illinois, USA. Springer.

[342] McMinn, P. and Holcombe, M. (2004). Hybridizing Evolutionary Testing with the Chaining Approach. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1363–1374, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[343] McMinn, P. and Holcombe, M. (2005). Evolutionary Testing of State-based Programs. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1013–1020, Washington, D.C., USA. ACM.

[344] McMinn, P. and Holcombe, M. (2006). Evolutionary Testing using an Extended Chaining Approach. *Evolutionary Computation*, **14**(1), 41–64.

[345] McMinn, P., Binkley, D., and Harman, M. (2005). Testability Transformation for Efficient Automated Test Data Search in the Presence of Nesting. In *Proceedings of the 3rd UK Software Testing Research Workshop (UKTest 2005)*, pages 165–182, Sheffield, UK. ACM.

[346] McMinn, P., Harman, M., Binkley, D., and Tonella, P. (2006). The Species per Path Approach to Search-based Test Data Generation. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis (ISSTA '06)*, pages 13–24, Portland, Maine, USA. ACM.

[347] McMinn, P., Binkley, D., and Harman, M. (2008). Empirical Evaluation of a Nesting Testability Transformation for Evolutionary Testing. *ACM Transactions on Software Engineering Methodology*.

[348] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**, 1087–1092.

[349] Michael, C. and McGraw, G. (1998). Automated Software Test Data Generation for Complex Programs. In *Proceedings of the 13th IEEE international conference on Automated software engineering (ASE '98)*, pages 136–146, Honolulu, Hawaii, USA. IEEE.

[350] Michael, C. C. and McGraw, G. E. (1997). Opportunism and Diversity in Automated Software Test Data Generation. Technical Report RSTR-003-97-13, Reliable Software Technologies.

[351] Michael, C. C., McGraw, G. E., Schatz, M. A., and Walton, C. C. (1997). Genetic Algorithms for Dynamic Test Data Generation. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering*, pages 307–308, Incline Village, NV, USA. IEEE Computer Society.

[352] Michael, C. C., McGraw, G. E., and Schatz, M. A. (1998). Opportunism and Diversity in Automated Software Test Data Generation. In *Proceedings of the 13th IEEE International Conference on Automated Software Engineering (ASE '98)*, pages 136–146, Hawaii, USA.

[353] Michael, C. C., McGraw, G., and Schatz, M. A. (2001). Generating Software Test Data by Evolution. *IEEE Transations on Software Engineering*, **27**(12), 1085–1110.

[354] Miller, J., Reformat, M., and Zhang, H. (2006). Automatic Test Data Generation using Genetic Algorithm and Program Dependence Graphs. *Information and Software Technology*, **48**(7), 586–605.

[355] Miller, W. and Spooner, D. L. (1976). Automatic Generation of Floating-Point Test Data. *IEEE Transactions on Software Engineering*, **2**(3), 223–226.

[356] ming Hsu, C. (2007). *A Test Data Evolution Strategy under Program Changes*. Master's thesis, National Sun Yat-sen University, Department of Information Management, Taiwan, China.

[357] Minohara, T. and Tohma, Y. (1995). Parameter Estimation of Hyper-Geometric Distribution Software Reliability Growth Model by Genetic Algorithms. In *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pages 324–329, Toulouse, France. IEEE Computer Society.

[358] Mitchell, B. S. (2002). *A Heuristic Search Approach to Solving the Software Clustering Problem*. Ph.D. thesis, Drexel University.

[359] Mitchell, B. S. and Mancoridis, S. (2002). Using Heuristic Search Techniques to Extract Design Abstractions from Source Code. In *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1375–1382, New York, USA. Morgan Kaufmann Publishers.

[360] Mitchell, B. S. and Mancoridis, S. (2003). Modeling the Search Landscape of Metaheuristic Software Clustering Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '03)*, pages 2499–2510, Chicago, IL, USA. Springer.

[361] Mitchell, B. S. and Mancoridis, S. (2006). On the Automatic Modularization of Software Systems using the Bunch Tool. *IEEE Transactions on Software Engineering*, **32**(3), 193–208.

[362] Mitchell, B. S. and Mancoridis, S. (2008). On the Evaluation of the Bunch Search-based Software Modularization Algorithm. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **12**(1), 77–93.

[363] Mitchell, B. S., Mancoridis, S., and Traverso, M. (2002). Search Based Reverse Engineering. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 431–438, Ischia, Italy. ACM.

[364] Mitchell, B. S., Mancoridis, S., and Traverso, M. (2004). Using Interconnection Style Rules to Infer Software Architecture Relations. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1375–1387, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[365] Moghadampour, G. (1999). *Using Genetic Algorithms in Testing a Distribution Protection Relay Software - A Statistical Analysis*. Ph.D. thesis, University of Vaasa, Department of Information Technology and Production Economics.

[366] Monnier, Y., Beauvais, J.-P., and Déplanche, A.-M. (1998). A Genetic Algorithm for Scheduling Tasks in a Real-Time Distributed System. In *Proceedings of the 24th EUROMICRO Conference (EUROMICRO '98)*, volume 2, pages 20708–20714, Vaesteraas, Sweden. IEEE Computer Society.

[367] Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions: I. Binary parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 178–187, Berlin. Springer.

[368] Nguyen, C., Miles, S., Perini, A., Tonella, P., Harman, M., and Luck, M. (2009). Evolutionary Testing of Autonomous Software Agents. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09)*, Budapest, Hungary.

[369] Nguyen, C. D., Perini, A., and Tonella, P. (2008). Constraint-based Evolutionary Testing of Autonomous Distributed Systems. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 221–230, Lillehammer, Norway. IEEE.

[370] Nisbet, A. (1998). GAPS: A Compiler Framework for Genetic Algorithm (GA) Optimised Parallelisation. In P. M. A. Sloot, M. Bubak, and L. O. Hertzberger, editors, *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking (HPCN '98)*, volume 1401 of *Lecture Notes In Computer Science*, pages 987–989, Amsterdam, Netherlands. Springer.

[371] O'Keeffe, M. and Ó Cinnéide, M. (2003). A Stochastic Approach to Automated Design Improvement. In *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java*, pages 59–62, Kilkenny City, Ireland. Computer Science Press, Inc.

[372] O'Keeffe, M. and Ó Cinnéide, M. (2004). Towards Automated Design Improvement Through Combinatorial Optimisation. In *Proceedings of the 26th International Conference on Software Engineering and Workshop on Directions in Software Engineering Environments (WoDiSEE '04)*, pages 75–82, Edinburgh, UK. ACM.

[373] O'Keeffe, M. and Ó Cinnéide, M. (2006). Search-based Software Maintenance. In *Proceedings of the Conference on Software Maintenance and Reengineering (CSMR '06)*, pages 249–260, Bari, Italy. IEEE Computer Society.

[374] O'Keeffe, M. and Ó Cinnéide, M. (2007). Getting the Most from Search-based Refactoring. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1114–1120, London, England. ACM.

[375] O'Keeffe, M. and Ó Cinnéide, M. (2008a). Search-based Refactoring: An Empirical Study. *Journal of Software Maintenance and Evolution: Research and Practice (Special Issue Search Based Software Engineering)*, **20**(5), 345–364.

[376] O'Keeffe, M. and Ó Cinnéide, M. (2008b). Search-based Refactoring for Software Maintenance. *Journal of Systems and Software*, **81**(4), 502–516.

[377] Pargas, R. P., Harrold, M. J., and Peck, R. R. (1999). Test-Data Generation using Genetic Algorithms. *The Journal of Software Testing, Verification and Reliability*, **9**(4), 263–282.

[378] Patton, R. M., Wu, A. S., and Walton, G. H. (2003). *A Genetic Algorithm Approach to Focused Software Usage Testing*, pages 259–286. Springer.

[379] Pedrycz, W. (2002). Computational Intelligence as an Emerging Paradigm of Software Engineering. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE '02)*, pages 7–14, Ischia, Italy. ACM.

[380] Pei, M., Goodman, E. D., Gao, Z., and Zhong, K. (1994). Automated Software Test Data Generation using a Genetic Algorithm. Technical report, Beijing University of Aeronautics and Astronautics.

[381] Pohlheim, H. and Wegener, J. (1999). Testing the Temporal Behavior of Real-Time Software Modules using Extended Evolutionary Algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and

R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '99)*, volume 2, page 1795, Orlando, Florida, USA. Morgan Kaufmann.

[382] Prutkina, M. and Windisch, A. (2008). Evolutionary Structural Testing of Software with Pointers. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 231–231, Lillehammer, Norway. IEEE.

[383] Räihä, O. (2008a). *Applying Genetic Algorithms in Software Architecture Design*. Master's thesis, Department of Computer Sciences, University of Tampere.

[384] Räihä, O. (2008b). *Genetic Synthesis of Software Architecture*. Ph.D. thesis, University of Tampere.

[385] Räihä, O. (2009). A Survey on Search-Based Software Design. Technical Report D-2009-1, Department of Computer Sciences University of Tampere.

[386] Räihä, O., Koskimies, K., and Mäkinen, E. (2008a). Genetic Synthesis of Software Architecture. In *Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL '08)*, LNCS 5361, pages 565–574, Melbourne, Australia. Springer.

[387] Räihä, O., Koskimies, K., Mäkinen, E., and Systä, T. (2008b). Pattern-Based Genetic Model Refinements in MDA. In *Proceedings of the Nordic Workshop on Model-Driven Engineering (NW-MoDE '08)*, pages 129–144, Reykjavik, Iceland. University of Iceland.

[388] Rajappa, V., Biradar, A., and Panda, S. (2008). Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory. In *Proceedings of the 1st International Conference on Emerging Trends in Engineering and Technology (ICETET '08)*, pages 298–303, Nagpur, India. IEEE.

[389] Reformat, M., Chai, X., and Miller, J. (2003). Experiments in Automatic Programming for General Purposes. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '03)*, pages 366–373, Sacramento, California, USA. IEEE Computer Society.

[390] Reformat, M., Chai, X., and Miller, J. (2007). On the Possibilities of (Pseudo-) Software Cloning from External Interactions. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, **12**(1), 29–49.

[391] Regehr, J. (2005). Random Testing of Interrupt-Driven Software. In *Proceedings of the 5th ACM International Conference on Embedded software*, pages 290–298, Jersey City, NJ, USA. ACM.

[392] Rela, L. (2004). *Evolutionary Computing in Search-based Software Engineering*. Master's thesis, Lappeenranta University of Technology.

[393] Ribeiro, J. C. B. (2008). Search-based Test Case Generation for Object-Oriented Java Software using Strongly-Typed Genetic Programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08) Workshop session: Graduate student workshops*, pages 1819–1822, Atlanta, GA, USA. ACM.

[394] Ribeiro, J. C. B., Zenha-Rela, M., and de Vega, F. F. (2007a). An Evolutionary Approach for Performing Structural Unit-Testing on Third-Party Object-Oriented Java Software. In *Proceedings of International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO '07)*, pages 379–388, Acireale, Italy. Springer.

[395] Ribeiro, J. C. B., Zenha-Rela, M., and de Vega, F. F. (2007b). eCrash: a Framework for Performing Evolutionary Testing on Third-Party Java Components. In *Proceedings of the I Jornadas sobre Algoritmos Evolutivos y Metaheuristicas (JAEM '07)*, pages 137–144, Zaragoza, Spain.

[396] Ribeiro, J. C. B., de Vega, F. F., and Zenha-Rela, M. (2007c). Using Dynamic Analysis of Java Bytecode for Evolutionary Object-Oriented Unit Testing. In *Proceedings of the 8th Workshop on Testing and Fault Tolerance (WTF '07)*, pages 143–156, Belm, Brazil.

[397] Ribeiro, J. C. B., Zenha-Rela, M. A., and de Vega, F. F. (2008a). A Strategy for Evaluating Feasible and Unfeasible Test Cases for the Evolutionary Testing of Object-Oriented Software. In *Proceedings of the 3rd international workshop on Automation of Software Test (AST '08)*, pages 85–92, Leipzig, Germany. ACM.

[398] Ribeiro, J. C. B., Zenha-Rela, M. A., and de Vega, F. F. (2008b). Strongly-Typed Genetic Programming and Purity Analysis: Input Domain Reduction for Evolutionary Testing Problems. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1783–1784, Atlanta, GA, USA. ACM.

[399] Roper, M. (1996). CAST with GAs - Automatic Test Data Generation via. Evolutionary Computation. In *Proceedings of IEE Colloquium on Computer Aided Software Testing Tools*, page 7, London, UK. IEE.

[400] Roper, M. (1997). Computer Aided Software Testing using Genetic Algorithms. In *Proceedings of the 10th International Software Quality Week*, pages 9T1–1–17, San Francisco, California, USA. Software Research Institute.

[401] Roper, M., Maclean, I., Brooks, A., Miller, J., and Wood, M. (1995). Genetic Algorithms and the Automatic Generation of Test Data. Technical report, Semin. Arthr. Rheum.

[402] Ryan, C. (2000). *Automatic Re-Engineering of Software using Genetic Programming*, volume 2. Kluwer Academic Publishers.

[403] Sagarna, R. (2007). *An Optimization Approach for Software Test Data Generation: Applications of Estimation of Distribution Algorithms and Scatter Search*. Ph.D. thesis, University of the Basque Country, San Sebastian, Spain.

[404] Sagarna, R. and Lozano, J. A. (2003). Variable Search Space for Software Testing. In *Proceedings of the IEEE International Conference on Neural Networks and Signal Processing (ICNNSP 2003)*, volume 1, pages 575–578, Nanjing, China. IEEE CS Press.

[405] Sagarna, R. and Lozano, J. A. (2005). On the Performance of Estimation of Distribution Algorithms applied to Software Testing. *Applied Artificial Intelligence*, **19**(5), 457–489.

[406] Sagarna, R. and Lozano, J. A. (2006). Scatter Search in Software Testing, Comparison and Collaboration with Estimation of Distribution Algorithms. *European Journal of Operational Research*, **169**(2), 392–412.

[407] Sagarna, R. and Lozano, J. A. (2007). Software Met-

rics Mining to Predict the Performance of Estimation of Distribution Algorithms in Test Data Generation. In A. L. C. Cotta, S. Reich and R. Schaefer, editors, *Knowledge-Driven Computing. Knowledge Engineering and Intelligent Computations*, volume 102/2008 of *Studies in Computational Intelligence*, pages 235–254. Springer-Verlag.

[408] Sagarna, R. and Lozano, J. A. (2008). Dynamic Search Space Transformations for Software Test Data Generation. *Computational Intelligence*, **24**(1), 23–61.

[409] Sagarna, R. and Yao, X. (2008). Handling Constraints for Search Based Software Test Data Generation. In *Proceedings of the 1st International Workshop on Search Based Software Testing (SBST)*, pages 232–240, Lillehammer, Norway. IEEE Computer Society.

[410] Sagarna, R., Arcuri, A., and Yao, X. (2007). Estimation of Distribution Algorithms for Testing Object Oriented Software. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '07)*, pages 438–444, Singapore. IEEE.

[411] Sahraoui, H. A., Valtchev, P., Konkobo, I., and Shen, S. (2002). Object Identification in Legacy Code as a Grouping Problem. In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment (COMPSAC '02)*, pages 689–696, Oxford, UK. IEEE.

[412] Saliu, M. O. and Ruhe, G. (2007). Bi-Objective Release Planning for Evolving Software Systems. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the Acm Sigsoft Symposium on the Foundations of Software Engineering*, pages 105–114, Dubrovnik, Croatia. ACM.

[413] Schneckenburger, C. and Schweiggert, F. (2008). Investigating the Dimensionality Problem of Adaptive Random Testing Incorporating a Search-based Testing Technique. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 241–250, Lillehammer, Norway. IEEE Computer Society.

[414] Schnier, T., Yao, X., and Liu, P. (2004). Digital filter design using multiple pareto fronts. *Soft Computing*, **8**(5), 332–343.

[415] Schoenauer, M. and Xanthakis, S. (1993). Constrained GA Optimization. In *Proceedings of the 5th International Conference on Genetic Algorithms (ICGA '93)*, pages 573–580, San Mateo, CA, USA. Morgan Kauffmann.

[416] Schultz, A. C., Grefenstette, J. J., and De Jong, K. A. (1993). Test and Evaluation by Genetic Algorithms. *IEEE Expert (also IEEE Intelligent Systems and Their Applications)*, **8**(5), 9–14.

[417] Schwefel, H. and Bäck, T. (1998). Artificial evolution: How and why? In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 1–19. John Wiley and Sons.

[418] Seesing, A. (2006). *EvoTest: Test Case Generation using Genetic Programming and Software Analysis*. Master's thesis, Delft University of Technology, Delft, The Netherlands.

[419] Seesing, A. and Groß, H.-G. (2006a). A Genetic Programming Approach to Automated Test Generation for Object-Oriented Software. *International Transactions on System Science and Applications*, **1**(2), 127–134.

[420] Seesing, A. and Groß, H.-G. (2006b). A Genetic Programming Approach to Automated Test Generation for Object-Oriented Software. Technical Report TUD-SERG-2006-017, Delft University of Technology.

[421] Seesing, A. and Groß, H.-G. (2006c). A Genetic Programming Approach to Automated Test Generation for Object-Oriented Software. In *Proceedings of the 1st International Workshop on Evaluation of Novel Approaches to Software Engineering*, Erfurt, Germany. NetObject Days 2006.

[422] Seng, O., Bauer, M., Biehl, M., and Pache, G. (2005). Search-based Improvement of Subsystem Decompositions. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1045–1051, Washington, D.C., USA. ACM.

[423] Seng, O., Stammel, J., and Burkhart, D. (2006). Search-based Determination of Refactorings for Improving the Class Structure of Object-Oriented Systems. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1909–1916, Seattle, Washington, USA. ACM.

[424] Shan, L. and Zhu, H. (2006). Testing Software Modelling Tools using Data Mutation. In *Proceedings of the 2006 International workshop on Automation of Software Test (AST '06)*, pages 43–49, Shanghai, China. ACM.

[425] Shan, Y., McKay, R. I., Lokan, C. J., and Essam, D. L. (2002). Software Project Effort Estimation Using Genetic Programming. In *Proceedings of the 2002 IEEE International Conference on Communications, Circuits and Systems and West Sino Expositions*, volume 2, pages 1108–1112, Chengdu, China. IEEE.

[426] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, **27**, 379–423 and 623–656.

[427] Sharma, V. S. and Jalote, P. (2008). Deploying Software Components for Performance. In *Proceedings of the 11th International Symposium on Component-Based Software Engineering (CBSE '08)*, pages 32–47, Karlsruhe, Germany. Springer.

[428] Shepperd, M. (2007). Software economics. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*, Los Alamitos, California, USA. IEEE Computer Society Press. This volume.

[429] Shepperd, M. J. (1995). *Foundations of software measurement*. Prentice Hall.

[430] Sheta, A. F. (2006). Estimation of the COCOMO Model Parameters Using Genetic Algorithms for NASA Software Projects. *Journal of Computer Science*, **2**(2), 118–123.

[431] Sheu, S.-T. and Chuang, Y.-R. (2006). A Pipeline-based Genetic Algorithm Accelerator for Time-Critical Processes in Real-Time Systems. *IEEE Transactions on Computers*, **55**(11), 1435–1448.

[432] Shukla, K. K. (2000). Neuro-Genetic Prediction of Software Development Effort. *Information and Software*

Technology, **42**(10), 701–713.

[433] Shyang, W., Lakos, C., Michalewicz, Z., and Schellenberg, S. (2008). Experiments in Applying Evolutionary Algorithms to Software Verification. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC '08)*, pages 3531–3536, Hong Kong, China. IEEE.

[434] Simons, C. L. and Parmee, I. C. (2006). Single and Multi-objective Genetic Operators in Object-oriented Conceptual Software Design. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1957–1958, Seattle, Washington, USA. ACM.

[435] Simons, C. L. and Parmee, I. C. (2007). A Cross-Disciplinary Technology Transfer for Search-based Evolutionary Computing: from Engineering Design to Software Engineering Design. *Engineering Optimization*, **39**(5), 631–648.

[436] Simons, C. L. and Parmee, I. C. (2008a). Agent-based Support for Interactive Search in Conceptual Software Engineering Design. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1785–1786, Atlanta, GA, USA. ACM.

[437] Simons, C. L. and Parmee, I. C. (2008b). User-centered, Evolutionary Search in Conceptual Software Design. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '08) (World Congress on Computational Inteligence)*, pages 869–876, Hong Kong, China. IEEE.

[438] Sinclair, M. C. and Shami, S. H. (1997). Evolving Simple Software Agents: Comparing Genetic Algorithm and Genetic Programming Performance. In *Proceedings of the Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA '97)*, pages 421–426, University of Strathclyde, Glasgow, UK. IEE.

[439] Snelting, G. (1998). Concept analysis — a new framework for program understanding. In *ACM SIGPLAN–SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'98)*, pages 1–10. Appears in SIGPLAN Notices 33(7):1–10.

[440] Sofokleous, A. A. and Andreou, A. S. (2007). Batch-Optimistic Test-Cases Generation Using Genetic Algorithms. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, volume 1, pages 157–164, Patras, Greece. IEEE Computer Society.

[441] Sofokleous, A. A. and Andreou, A. S. (2008a). Automatic, Evolutionary Test Data Generation for Dynamic Software Testing. *Journal of Systems and Software*, **81**(11), 1883–1898.

[442] Sofokleous, A. A. and Andreou, A. S. (2008b). Dynamic Search-based Test Data Generation Focused on Data Flow Paths. In *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS '08)*, pages 27–35, Barcelona, Spain.

[443] Stephenson, M., Amarasinghe, S., Martin, M., and O'Reilly, U.-M. (2003). Meta Optimization: Improving Compiler Heuristics with Machine Learning. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming Language Design and Implementation (PLDI '03)*, pages 77–90, San Diego, California, USA. ACM.

[444] Sthamer, H., Baresel, A., and Wegener, J. (2001). Evolutionary Testing of Embedded Systems. In *Proceedings of the 14th International Internet & Software Quality Week (QW '01)*, pages 1–34, San Francisco, California, USA.

[445] Sthamer, H.-H. (1995). *The Automatic Generation of Software Test Data Using Genetic Algorithms*. Ph.D. thesis, University of Glamorgan.

[446] Su, S., Zhang, C., and Chen, J. (2007). An Improved Genetic Algorithm for Web Services Selection. In *Proceedings of the 7th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS '07)*, volume 4531 of *LNCS*, pages 284–295, Paphos, Cyprus. Springer.

[447] Sutton, A., Kagdi, H., Maletic, J. I., and Volkert, L. G. (2005). Hybridizing Evolutionary Algorithms and Clustering Algorithms to Find Source-Code Clones. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1079–1080, Washington, D.C., USA. ACM.

[448] Tang, M. and Dong, J. (2005). Simulated annealing genetic algorithm for surface intersection. In *Advances in Natural Computation*, volume 3612 of *Lecture Notes in Computer Science*, pages 48–56. Springer.

[449] Tlili, M., Wappler, S., and Sthamer, H. (2006). Improving Evolutionary Real-Time Testing. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1917–1924, Seattle, Washington, USA. ACM.

[450] Tonella, P. (2004). Evolutionary Testing of Classes. In *Proceedings of the 2004 ACM SIGSOFT Iinternational Symposium on Software Testing and Analysis (ISSTA '04)*, pages 119–128, Boston, Massachusetts, USA. ACM.

[451] Tracey, N., Clark, J., and Mander, K. (1998a). Automated Program Flaw Finding using Simulated Annealing. In *Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '98)*, pages 73–81, Clearwater Beach, Florida, USA. ACM.

[452] Tracey, N., Clark, J., and Mander, K. (1998b). The Way Forward for Unifying Dynamic Test-Case Generation: the Optimisation-based Approach. In *Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications (DCIA '98)*, pages 169–180, Johannesburg, South Africa. University of the Witwatersrand.

[453] Tracey, N., Clark, J., Mander, K., and McDermid, J. (2000). Automated Test-Data Generation for Exception Conditions. *Software Practice and Experience*, **30**(1), 61–79.

[454] Tracey, N., Clark, J., McDermid, J., and Mander, K. (2002). A Search-based Automated Test-Data Generation Framework for Safety-Critical Systems. In P. Henderson, editor, *Systems engineering for business process change: new directions*, pages 174–213. Springer-Verlag New York, Inc., New York, NY, USA.

[455] Tracey, N. J. (2000). *A Search-based Automated Test-Data Generation Framework for Safety-Critical Software*.

Ph.D. thesis, University of York.

[456] Tsai, W., Zhang, D., Paul, R., and Chen, Y. (2005). Stochastic Voting Algorithms for Web Services Group Testing. In *Proceedings of the 5th International Conference on Quality Software (QSIC '05)*, pages 99–108, Melbourne, Australia. IEEE Computer Society.

[457] Uyar, H. T., Uyar, A. S., and Harmanci, E. (2006). Pairwise Sequence Comparison for Fitness Evaluation in Evolutionary Structural Software Testing. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1959–1960, Seattle, Washington, USA. ACM.

[458] Van Belle, T. and Ackley, D. H. (2002). Code Factoring and the Evolution of Evolvability. In *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO '02)*, pages 1383–1390, New York, USA. Morgan Kaufmann Publishers.

[459] Vijayalakshmi, K., Ramaraj, N., and Amuthakkannan, R. (2008). Improvement of Component Selection Process using Genetic Algorithm for Component-Based Software Development. *International Journal of Information Systems and Change Management*, **3**(1), 63–80.

[460] Vivanco, R. and Jin, D. (2008). Enhancing Predictive Models using Principal Component Analysis and Search Based Metric Selection: A Comparative Study. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08)*, pages 273–275, Kaiserslautern, Germany. ACM.

[461] Vivanco, R. and Pizzi, N. (2004). Finding Effective Software Metrics to Classify Maintainability Using a Parallel Genetic Algorithm. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1388–1399, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[462] Vivanco, R. A. and Jin, D. (2007). Selecting Object-Oriented Source Code Metrics to Improve Predictive Models using a Parallel Genetic Algorithm. In *Proceedings of Conference on Object Oriented Programming Systems Languages and Applications Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 769–770, Montreal, Quebec, Canada. ACM.

[463] Waeselynck, H., Thévenod-Fosse, P., and Abdellatif-Kaddour, O. (2007). Simulated Annealing Applied to Test Generation: Landscape Characterization and Stopping Criteria. *Empirical Software Engineering*, **12**(1), 35–63.

[464] Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., and Roos, R. S. (2006). Time-Aware Test Suite Prioritization. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis (ISSTA '06)*, pages 1–12, Portland, Maine, USA. ACM.

[465] Wang, H.-C. (2006). *A Hybrid Genetic Algorithm for Automatic Test Data Generation*. Master's thesis, National Sun Yat-sen University, Department of Information Management, Taiwan, China.

[466] Wang, Y., Bai, Z., Zhang, M., Du, W., Qin, Y., and Liu, X. (2008a). Fitness calculation approach for the switch-case construct in evolutionary testing. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1767–1774, Atlanta, GA, USA. ACM.

[467] Wang, Z., Tang, K., and Yao, X. (2008b). A Multi-Objective Approach to Testing Resource Allocation in Modular Software Systems. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC '08)*, pages 1148–1153, Hong Kong, China. IEEE.

[468] Wappler, S. (2004). *Using Evolutionary Algorithms for the Test of Object-Oriented Systems*. Master's thesis, University of Potsdam.

[469] Wappler, S. (2008). *Automatic Generation of Object-Oriented Unit Tests using Genetic Programming*. Ph.D. thesis, Technical University of Berlin.

[470] Wappler, S. and Lammermann, F. (2005). Using Evolutionary Algorithms for the Unit Testing of Object-Oriented Software. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1053–1060, Washington, D.C., USA. ACM.

[471] Wappler, S. and Schieferdecker, I. (2007). Improving Evolutionary Class Testing in the Presence of Non-Public Methods. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*, pages 381–384, Atlanta, Georgia, USA. IEEE.

[472] Wappler, S. and Wegener, J. (2006a). Evolutionary Unit Testing Of Object-Oriented Software Using A Hybrid Evolutionary Algorithm. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC '06)*, pages 851–858, Vancouver, BC, Canada. IEEE.

[473] Wappler, S. and Wegener, J. (2006b). Evolutionary Unit Testing of Object-Oriented Software using Strongly-Typed Genetic Programming. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1925–1932, Seattle, Washington, USA. ACM.

[474] Wappler, S., Baresel, A., and Wegener, J. (2007). Improving Evolutionary Testing in the Presence of Function-Assigned Flags. In *Proceedings of 2nd Testing: Academic & Industrial Conference - Practice and Research Techniques (TAICPART-MUTATION '07)*, pages 23–34, Windsor, UK. IEEE.

[475] Watkins, A. and Hufnagel, E. M. (2006). Evolutionary Test Data Generation: A Comparison of Fitness Functions. *Software: Practice and Experience*, **36**(1), 95–116.

[476] Watkins, A., Berndt, D. J., Aebischer, K., Fisher, J. W., and Johnson, L. (2004). Breeding Software Test Cases for Complex Systems. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS '04)*, Hawaii, USA. IEEE Computer Society.

[477] Watkins, A., Hufnagel, E. M., Berndt, D. J., and Johnson, L. (2006). Using Genetic Algorithms and Decision Tree Induction to Classify Software Failures. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, **16**(2), 269–291.

[478] Watkins, A. L. (1995). The Automatic Generation of Test Data using Genetic Algorithms. In *Proceedings of the 4th Software Quality Conference*, volume 2, pages 300–309,

Dundee, UK. ACM.

[479] Wegener, J. and Bühler, O. (2004). Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1400–1412, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[480] Wegener, J. and Grochtmann, M. (1998). Verifying Timing Constraints of Real-Time Systems by means of Evolutionary Testing. *Real-Time Systems*, **15**(3), 275–298.

[481] Wegener, J. and Mueller, F. (2001). A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. *Real-Time Systems*, **21**(3), 241–268.

[482] Wegener, J., Sthamer, H., Jones, B. F., and Eyres, D. E. (1997a). Testing Real-Time Systems using Genetic Algorithms. *Software Quality*, **6**(2), 127–135.

[483] Wegener, J., Grochtmann, M., and Jones, B. (1997b). Testing Temporal Correctness of Real-Time Systems by Means of Genetic Algorithms. In *Proceedings of the 10th International Software Quality Week (QW '97)*, San Francisco, California, USA.

[484] Wegener, J., Baresel, A., and Sthamer, H. (2001). Evolutionary Test Environment for Automatic Structural Testing. *Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms*, **43**(14), 841–854.

[485] Wegener, J., Buhr, K., and Pohlheim, H. (2002a). Automatic Test Data Generation for Structural Testing of Embedded Software Systems by Evolutionary Testing. In *Proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO '02)*, pages 1233–1240, New York, USA. Morgan Kaufmann Publishers Inc.

[486] Wegener, J., Baresel, A., and Sthamer, H. (2002b). Suitability of Evolutionary Algorithms for Evolutionary Testing. In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment (COMPSAC '02)*, pages 287 – 289, Oxford, UK. IEEE Computer Society.

[487] Wen, F. and Lin, C.-M. (2008). Multistage Human Resource Allocation for Software Development by Multiobjective Genetic Algorithm. *The Open Applied Mathematics Journal*, **2**, 95–103.

[488] White, D. R., Clark, J., Jacob, J., and Poulding, S. M. (2008). Searching for Resource-Efficient Programs: Low-Power Pseudorandom Number Generators. In M. Keijzer, editor, *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pages 1775–1782, Atlanta, GA, USA. ACM.

[489] Whitley, D., Sutton, A. M., and Howe, A. E. (2008). Understanding elementary landscapes. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO 2008)*, pages 585–592, New York, NY, USA. ACM.

[490] Williams, K. P. (1998). *Evolutionary Algorithms for Automatic Parallelization*. Phd thesis, University of Reading, UK, Department of Computer Science.

[491] Windisch, A. (2008). Search-based Testing of Complex Simulink Models Containing Stateflow Diagrams. In *Proceedings of 1st International Workshop on Search-Based Software Testing (SBST) in conjunction with ICST 2008*, pages 251–251, Lillehammer, Norway. IEEE Computer Society.

[492] Windisch, A., Wappler, S., and Wegener, J. (2007). Applying Particle Swarm Optimization to Software Testing. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1121–1128, London, England. ACM.

[493] Xanthakis, S., Ellis, C., Skourlas, C., Le Gall, A., Katsikas, S., and Karapoulios, K. (1992). Application of Genetic Algorithms to Software Testing. In *Proceedings of the 5th International Conference on Software Engineering and Applications*, pages 625–636, Toulouse, France.

[494] Xiao, J., Lam, C. P., Li, H., and Wang, J. (2006). Reformulation of the Generation of Conformance Testing Sequences to the Asymmetric Travelling Salesman Problem. In *Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation (GECCO '06)*, pages 1933–1940, Seattle, Washington, USA. ACM.

[495] Xiao, M., El-Attar, M., Reformat, M., and Miller, J. (2007). Empirical Evaluation of Optimization Algorithms when used in Goal-oriented Automated Test Data Generation Techniques. *Empirical Software Engineering*, **12**(2), 183–239.

[496] Xie, X., Xu, B., Shi, L., Nie, C., and He, Y. (2005a). A Dynamic Optimization Strategy for Evolutionary Testing. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC '05)*, pages 568–575, Taibei, Taiwan. IEEE Computer Society.

[497] Xie, X., Xu, B., Nie, C., Shi, L., and Xu, L. (2005b). Configuration Strategies for Evolutionary Testing. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC '05)*, pages 13–14, Edinburgh, UK. IEEE Computer Society.

[498] Yang, L., Jones, B. F., and Yang, S.-H. (2006). Genetic Algorithm based Software Integration with Minimum Software Risk. *Information and Software Technology*, **48**(3), 133–141.

[499] Yoo, S. and Harman, M. (2007). Pareto Efficient Multi-Objective Test Case Selection. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07)*, pages 140–150, London, England. ACM.

[500] Zhan, Y. and Clark, J. A. (2004). Search Based Automatic Test-Data Generation at an Architectural Level. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO '04)*, volume 3103/2004 of *Lecture Notes in Computer Science*, pages 1413–1424, Seattle, Washington, USA. Springer Berlin / Heidelberg.

[501] Zhan, Y. and Clark, J. A. (2005). Search-based Mutation Testing for Simulink Models. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, pages 1061–1068, Washington, D.C., USA. ACM.

[502] Zhan, Y. and Clark, J. A. (2006). The State Problem for Test Generation in Simulink. In *Proceedings of the 8th*

annual Conference on Genetic and Evolutionary Computation (GECCO '06), pages 1941–1948, Seattle, Washington, USA. ACM.

[503] Zhan, Y. and Clark, J. A. (2008). A Search-based Framework for Automatic Testing of MATLAB/Simulink Models. *Journal of Systems and Software*, **81**(2), 262–285.

[504] Zhang, C., Su, S., and Chen, J. (2006). A Novel Genetic Algorithm for QoS-Aware Web Services Selection. In *Proceedings of the 2nd International Workshop on Data Engineering Issues in E-Commerce and Services (DEECS '06)*, volume 4055 of *LNCS*, pages 224–235, San Francisco, CA, USA. Springer.

[505] Zhang, C., Su, S., and Chen, J. (2007a). DiGA: Population diversity handling genetic algorithm for QoS-aware web services selection. *Computer Communications*, **30**(5), 1082–1090.

[506] Zhang, H. (2004). *Automatic Test Data Generation using Genetic Algorithm and Program Dependence Graphs*. Master's thesis, University of Alberta (Canada).

[507] Zhang, X., Meng, H., and Jiao, L. (2005). Intelligent particle swarm optimization in multiobjective optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 714–719, Edinburgh, UK. IEEE Press.

[508] Zhang, Y., Harman, M., and Mansouri, S. A. (2007b). The Multi-Objective Next Release Problem. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO '07)*, pages 1129–1137, London, UK. ACM.

[509] Zhang, Y., Finkelstein, A., and Harman, M. (2008). Search Based Requirements Optimisation: Existing Work & Challenges. In *Proceedings of the 14th International Working Conference, Requirements Engineering: Foundation for Software Quality (REFSQ '08)*, volume 5025 of *LNCS*, pages 88–94, Montpellier, France. Springer.

[510] Zhong, H., Zhang, L., and Mei, H. (2008). An Experimental Study of Four Typical Test Suite Reduction Techniques. *Information and Software Technology*, **50**(6), 534–546.

View publication stats